ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincerest thanks to my thesis supervisor, Dr. Terrill Fancott, who outlined the philosophy and direction of this research and read the many drafts of this thesis, making numerous suggestions for its improvement.

I would like to thank Jacques Blaison who designed and assembled the hardware for the experiment. His technical expertise, cooperation, and help were invaluable.

To friends, who through their encouragement, moral support, and interest saw me through this thesis, I say thank you. You cannot know how much you have meant to me. This includes two people who are very special to me, my parents, to whom thank you can never be enough for their years of love and caring.

I dedicate this thesis to Anatoli Shcharansky, a political prisoner of the U.S.S.R and fellow computer scientist, with the fervent hope, next year in Jerusalem.

TABLE OF CONTENTS

Page

SIGNATURE PAGE..... ii

ABSTRACT..... iii

ACKNOWLEDGEMENTS..... iv

TABLE OF CONTENTS..... v

LIST OF FIGURES.....viii

CHAPTER

I. INTRODUCTION 1

1.1 THE ENVIRONMENT OF COMPUTER ARCHITECTURE..... 1

1.2 TECHNOLOGY AND THE DESIGN PROCESS..... 2

1.3 PHILOSOPHY BEHIND THE THESIS..... 6

1.4 SCOPE OF THESIS..... 7

1.4.1 CHOICE AND LIMITATIONS IN HARDWARE..... 8

1.4.2 DECISIONS IN SOFTWARE..... 9

II. OVERVIEW 14

2.1 TECHNOLOGICAL ADVANCES AND PREDICTIONS..... 14

2.2 DESIGN OF MODULAR HARDWARE SYSTEMS..... 16

2.3 A COMPUTER MODULE (CM) AS A DESIGN UNIT..... 20

2.4 HARDWARE INTERGONNECTION STRUCTURES/TOPOLOGY..... 23

2.5 OPERATING SYSTEM DESIGN..... 30

2.5.1 STRUCTURE, SYNCHRONIZATION AND COMMUNICATION... 31

CHAPTER Page

2.5.2 NETWORK OPERATING SYSTEMS..... 42

2.5.3 OPERATING SYSTEMS TO FUNCTION MODULES..... 45

III. MMCMS EXPERIMENT 47

3.1 DESCRIPTION OF THE TI ENVIRONMENT..... 47

3.1.1 ARCHITECTURE OF THE TI980..... 48

3.1.2 OPERATING SYSTEM OF THE TI980..... 51

3.1.3 FUNCTIONS OF THE CONSOLE DSR..... 52

3.1.4 COMMUNICATION AND CONTROL FLOW..... 56

3.2 DESCRIPTION OF THE SMS300..... 60

3.2.1 ARCHITECTURE OF THE SMS300..... 60

3.2.2 INSTRUCTION SET OF THE SMS300..... 64

3.2.3 INSTRUCTION SET AND PROGRAMMING..... 65

3.3 COMMUNICATION IN THE EXPERIMENTAL SYSTEM..... 68

3.3.1 COMMUNICATION PROTOCOL..... 68

3.3.2 MESSAGE FORMAT..... 71

3.4 IMPLEMENTATION DETAILS IN TI HOST..... 74

3.5 IMPLEMENTATION DETAILS OF SMS FUNCTION MODULE.... 77

3.6 DISTRIBUTED CONTROL FLOW..... 79

3.7 PROGRAM DEVELOPMENT AND LOADING..... 82

3.7.1 HARDWARE DETAILS..... 84

3.7.2 DISPLAY AND CONTROL..... 85

CHAPTER Page

IV. ANALYSIS OF MMCMS EXPERIMENT 87

4.1 MESSAGE-BASED COMMUNICATION..... 87

4.2 RESULTS AND APPLICABILITY..... 90

4.3 CONCLUSIONS AND FURTHER WORK..... 92

ANNOTATED BIBLIOGRAPHY 97

APPENDICES 110

APPENDIX A - TI AND SMS BACKGROUND

APPENDIX B - HARDWARE DETAILS OF THE EXPERIMENT

APPENDIX C - MESSAGE FORMATS

APPENDIX D - DEVELOPMENT SYSTEM PROGRAMS

APPENDIX E - EXPERIMENTAL MMCMS SYSTEM PROGRAMS

LIST OF FIGURES

Figure		Page
2.1	Hardware Interconnection Taxonomy	24
3.1	TI980B Architecture in PMS Notation	50
3.2	TI980B System Block Diagram	50
3.3	Interrelationship among Calling Program, INTRIP, DSRs, and System Tables	53
3.4	Control Flow for Function at the Operating System Level	57
3.5	Control Flow for Function Provided as a Utility	61
3.6	SMS300 System Block Diagram	63
3.7	SMS300 Architecture in PMS Notation	63
3.8	Multilevel Protocol for Message Transfer	70
3.9	HII/Heading Item Indicator Coding	73
3.10	Message Format	73
3.11	Implementation of Communication Routine	74
3.12	Program Control Flow with Distributed Software	80
3.13	Physical Setup of Experimental/Development System in PMS Notation	86
4.1	Memory Requirements for Programs	91
4.2	Message Communication Base Layer	93
4.3	Control Flow with JCMAIN Distributed	95

I. INTRODUCTION

1.1 THE ENVIRONMENT OF COMPUTER ARCHITECTURE

Computers have evolved through four generations since their inception in the 1940's, each time due primarily to technological advances in electronic circuitry (Weitzman, 1974). Starting from discrete technology, circuits have moved from small, to medium, to large scale integration of components, enabling more electronics and more capability to be compacted into the small area of one silicon chip. As well, manufacturing costs have decreased (Noyce, 1977), (Joseph, 1976). These improvements have had a great impact on computer systems. They have not only reduced component costs but have significantly altered the hardware design process. The availability of sophisticated functions in integrated circuit packages can reduce the process of system design to the fitting together of standardized chips. The testing of such systems is simplified because the correct functioning of the integrated circuit chips has been ensured by the manufacturer.

There is a limit, however, to the applicability of this progression to classical computer architecture. When integration provides more capability and intelligence per chip or module than the identifiable components of computer structure, the structure itself must evolve. The

contemporary microcomputer provides a building block above this level, and computer architects are attempting to adapt the concepts of computer hardware and software accordingly. The solution is more complex than the straightforward interconnection of a number of microprocessors. The following section will review the background and interactions of technology, computer architecture, and operating systems.

1.2 TECHNOLOGY AND THE DESIGN PROCESS

The design of an architecture for a computer system and its instruction set requires an intuitive feel for the interactions among the various disciplines of design, electronics, hardware, software, and marketing. Furthermore, design is an iterative process, with the computer architect producing a hardware/software complex subject to realistic constraints, and then modifying and upgrading it in response to feedback from system programmers, results of application benchmarks, and experience gained on the new machine (Bell, 1975). Decisions have always been made whether a particular system function should be implemented in hardware, firmware, or software. Technological advances in LSI have helped tip the balance in favour of more sophisticated functions in hardware. Some examples are floating point arithmetic, Fast Fourier Transform analysis, data type conversion, call and

exit of subprograms, and dynamic memory allocation (Sell, 1975). In fact, to investigate the limits of hardware, an entire computer complex, SYMBOL-2R, has been constructed (Smith, 1971), (Richards, 1975). It demonstrates the feasibility of a high-level language, virtual memory, timesharing system that operates entirely without system software. A hardware supervisor coordinates a set of special purpose processors and multiplexes tasks among them from up to 31 users. The I/O processor and channel controller handle the loading and editing of programs and user I/O. A separate translator processor accepts the user programs in character string form and produces object code that runs on the central processor.

To be effective in attaining an integrated design of hardware, software and firmware, one must understand the issues and alternatives in designing a computer system. Too often, the design is weakened when a tradeoff is made to improve performance or simplify hardware, leaving programmers to live with the inconsistencies (Allison, 1976). Specifically, today's computers do not support the programming of operating systems very well and considerable complexity is introduced to cope with inadequacies inherent in the hardware (Liskov, 1972).

Progress in computers and technology has made new choices available to the computer architect-designer. Microprogrammable computers allow the flexible definition of

operating system functions at the elementary operation level. Much research is being done to investigate the criteria for determining which functions are appropriate for implementation in microcode (Sockhut, 1975), (Perez, 1975), (Cox, 1974), (Brown, 1976), (Rosen, 1968). A wide choice of supervisory functions that are part of operating systems are possible candidates, for example, synchronization primitives, memory management, and error control functions. This is not far-fetched, as many of the more advanced features of present day processors that are taken for granted, such as index registers and stacks, represent exactly such tradeoffs (Mandell, 1972).

Another possibility is the implementation of operating system functions in dedicated computer modules (Falk, 1974), (Teichholtz, 1975), (Arden, 1975), (Fancott, 1977), (Poujoulat, 1977). This is a logical extension of intelligent peripherals, where functions provided by software on the central processor have migrated to a microprocessor in each peripheral. Operating systems made up of structured, modular software can be partitioned along functional lines to reside in separate modules. At the present time, the low cost and availability of microcomputers (some already integrated onto one chip), with significant computational and control processing ability, make them ideal modules. What is required is that each function have large independent processing times as compared

to their intercommunication requirements. As functions at the operating system level become more standardized and technology takes the next leap in integration, dedicated function modules will be able to be designed specifically for software tasks. With the advent of Very Large Scale Integration (VLSI) of components, designers will be able to replace programs by placing a full application on a single chip (Joseph, 1972). Computer modules (CMs) (Bell, 1973), as the building blocks of a system can be used as hardware modules dedicated to operating system functions, whereby the system is defined physically and functionally at the same time. This implies that the functions of the operating system and its distribution are defined concurrently with the architecture (Poujoulat, 1977), (Joseph, 1976).

Another advantage to building with CMs is that the architect-designer can put together a system in a structured manner, a module at a time. If a set of CMs with different features and capabilities exists then the designer will be able to choose the most appropriate for the function required. Moreover, CMs can be interconnected in a way that parallels and is best suited to the application (Raphael, 1975), (Joseph, 1974).

1.3 PHILOSOPHY BEHIND THE THESIS

Computer systems can be interconnected in a variety of ways, from tightly-coupled multiprocessor systems to packet-switching networks of independent and often different computer systems. Each processor or node can be viewed as a function module, that is as a module capable of providing one or more functions to the end user.

The system can be analyzed at two distinct levels. The hardware base layer is made up of a set of processors and their physical links. The architecture of each processor, its connection to others, and the topology of the interconnection can all be examined. The software communication layer is made up of a set of routines which communicate and synchronize with one another according to a given protocol. The mechanism for communication, the techniques to guarantee mutual exclusion, and the communication protocol define a logical path among the processors and software. Both these levels are examined in detail in the next chapter. Since the system is modular and well-defined, cooperation and synchronization among processors can be ensured using the rules governing the interaction of processes in an operating system (Dijkstra, 1968), (Brinch Hansen, 1970), (Habermann, 1972).

A message-based communication system is proposed since it has been shown to be appropriate for both an operating

system on one processor, as well as a system distributed over a network of processors (Brinch Hansen, 1971), (Spier, 1969), (Walden, 1972), (Ruschitzka, 1973). Other researchers have recommended that scaling down protocols appropriate to a full-fledged network will aid in structuring and clarify the interaction of operating system functions (Metcalfe, 1972), (Fuller, 1973a), (Probst, 1977). The design of a communication facility based on the explicit exchange of messages is presented, including one possible message format and a protocol within an existing minicomputer operating system. This facility was implemented in an experimental system to investigate the complexities and problems in distributed communication.

1.4 SCOPE OF THESIS

The design of a communications function for a distributed system is reported. This includes an experimental implementation of the function itself and its interface to an existing operating system. Redesign of the operating system to take full advantage of the distributed architecture is not considered here.

Minimally, two interconnected function modules were required to implement the simplest distributed function module architecture and to enable the experimental work to proceed. A basic operating system was examined and one

operating system function chosen for the function module. This was implemented as a series of assembler language programs in a microcomputer. A message format and protocol were developed, which in conjunction with communication software in each module define a logical path between the two interconnected computers.

1.4.1 CHOICE AND LIMITATIONS IN HARDWARE

The microprocessor chosen as the test candidate for the function module was the Scientific Micro Systems 300. The SMS 300 was selected over a number of more popular, widely used micros for its capabilities - bit and byte addressability in both memory and on the bus, and high speed operation (300 ns instruction time) - that are appropriate to a module providing an I/O function. As well, the SMS meets the criteria for good architectural design as outlined in Bell's paper (1975) and its Interface Vector provides an implementation of the 'ports' structure of a CM module (Bell, 1973). All other operating system functions were left intact, provided by a single user, non interrupt driven operating system that executes on a Texas Instrument model 980B minicomputer.

In the current implementation, the physical connection is achieved by connecting together the TI I/O bus and the SMS Interface Vector. A 1K byte memory has been inserted

between the two buses to simplify arbitration logic and remove timing and electrical considerations. The memory proved to be invaluable during the testing phase because it provided a recent history of message transfers. Limitations in the hardware configuration of the TI and SMS architecture precluded direct memory access (DMA) transfer.

Because there is no interval timer on either the TI980 or SMS to control the execution of programs, each program is allowed to continue until it voluntarily gives up the central processor. This also implied that programs waiting for a message, could not use a timeout as a criterion for requesting a retransmission. It should be reiterated that it was only hardware limitations that dictated the specific experimental configuration. As will be seen in the next section, the software is much more general, and in fact, will map onto various hardware connections.

1.4.2 DECISIONS IN SOFTWARE

The operating system used for the implementation was the Texas Instrument's basic operating system - a single user, non interrupt driven system. The nature of this system imposed some restrictions on the details of implementation since it is incapable of supporting concurrency or processes. The TI system was sufficient to enable the experimental implementation and testing of the

communications function.

To isolate the problems involved in communication between modules, the extent of distribution of the operating system functions has been limited to the implementation of a single remote function, the console log device service routine (DSR). This choice has the advantage of providing a device capable of visually displaying both input and output which is useful during testing to demonstrate the results. The DSR level meets the requirements for a function module because it performs a well-defined set of operations. It can be moved with minimum alteration to the existing operating system and can be readily integrated with the peripheral device. As well, the DSR's removal can save processing time on the main CPU and data traffic on the I/O bus. As well, it can be moved with minimum alteration to the existing operating system. It is envisioned that the work reported here will be continued with a second level distribution of command formats, cracking, and error handling from the operator communication package. Suggestions for facilities that can be implemented at this level can be found in the literature on intelligent terminals (Gray, 1975), (Whiting, 1975), (Dromard, 1974), (Alsberg, 1976). The assumption is that the addition of more intelligence at the periphery will not raise the cost of the interface by any significant amount. At the same time, a considerable load will be removed from the central

processor and the data traffic between the two computers will be reduced.

It was decided to use the operating system with few modifications and, in fact, to make these as transparent to the user as possible. Uniprogramming has the advantage of restricting the scope of the problem to a reasonable size. This is in line with the learning process recommended by Stoy (1972), that one learn to build good operating systems for single users before trying to satisfy many users simultaneously. Many of the functions necessary in a multiprogramming system must be provided in a single user operating system as well. The decisions about how to partition the system, which functions to distribute and how this is to be done are just as applicable to a single user system. The simplicity of the operating system enables concentration on the development of a standard intercommunication philosophy, expressed in the form of a common interface specification and a well-defined protocol.

Conceptually, two processes communicate by explicitly exchanging messages via an interconnecting link. As part of the interface requirements, each hardware module connected to the link must contain a communication process that implements a send and receive function. The send function enables the module to put messages which it wants delivered to other modules onto the link. The receive function is able to read the address field of all messages passing its

station. If the name specified as the destination matches its own name, the receiver copies the message into a local buffer. In this implementation, each computer has a communication routine that enables it to send and receive messages and, in this way, meet the specifications. Only the I/O utility at the lowest level is aware of the physical characteristics of the link, and it shields even the communication routine from the hardware interface. This implies that the layer responsible for communication is mappable onto various hardware connections.

In a single user, sequential processing system, synchronization is inherent, as each program executes to completion in a predetermined order. In an environment that implements processes, some method of synchronization among them must also be provided. The problem is greater when processes may be implemented in a distributed system. With multiple processors, the exchange of messages can provide the necessary synchronization. If the sender cannot continue until the function requested has been performed, it waits for a return message from the destination process. All that is required is the guarantee that no process can access a message before it is complete or acquire a buffer before it has been emptied. This is the classic producer-consumer problem (Tsichritzis, 1974). Mutual exclusion to a message can be provided by hardware, implemented by arbitration logic on the link. Since in this

implementation messages reside in a memory accessible by two processors, a hardware interlock is required to guarantee mutual exclusion. This has been implemented as a 1 byte flag in the message buffer which is updated indivisibly when the message is complete and can be accessed safely.

II. OVERVIEW

2.1 TECHNOLOGICAL ADVANCES AND PREDICTIONS

Since its inception, semiconductor technology has consistently made remarkable progress. Today, a chip one quarter inch square can house 2^{18} electronic elements, more than the most complex piece of equipment that could be built in 1950 (Noyce, 1977). The complexity of the integrated circuit on a chip has doubled every year without any significant slowdown predicted (Moore's Law). The result is that smaller electronic components are able to perform increasingly complex functions at ever higher speeds, with improved reliability (Hodges, 1976) and at ever lower costs.

The computer industry has been one beneficiary of these advances in integration. Semiconductor memories have replaced core storage as the standard because they provide faster cycle times, require less power, and take up less physical space. More sophisticated logic elements in single IC packages are radically affecting hardware design. As this process continues, the functions that can be provided become more complex, but tend also to have less application potential, unless they can be programmed by the designer. This has led to a variety of microprocessors on single chips becoming commercially available. Combined with the memory

chips, programmed logic is being used to replace hardwired random logic for many applications (Lee, 1974). Their availability at low cost is having a significant impact on computer architectures.

It is interesting to examine predictions made within the last decade for the capability of a computer on a chip. In 1972, it was thought that within twenty-five years, a microcomputer could be developed that would execute ten million instructions per second and cost about a dollar. The system would be modest, consisting of 16K words of 32 bits each, byte addressable, a small but carefully chosen instruction set, a few general purpose registers and input/output through at least one of these registers (Foster, 1972). Other than a large word length, the ideas presented have turned out to be conservative (Foster admitted that to predict as far as 25 years into the future was virtually impossible).

Within two years, it was predicted that a standalone personal machine with communications adaptor to access a large, remote data base was technologically, and economically feasible within six years - by the year 1980. This computer would have a higher level language (BASIC was suggested) implemented in a 40 pin dual inline package, 8 additional chips to give 128K bytes memory, another dozen or so packages for control, and a floppy disc for secondary storage (Lee, 1974).

A year later, there was a prediction made that a single package consisting of several LSI processors was possible that would yield data processing power of 10 million instructions per second (MIPS), and LSI storage of 10^9 bits, and an archival storage of 10^{12} bits. This compares with ten IBM S370/158's with 4 megabytes storage and more than 200 reels of tape on-line (Greenblott, 1975)- a staggering display of power and capability.

2.2 DESIGN OF MODULAR HARDWARE SYSTEMS

The continuing increase in integration of LSI circuits is creating an environment where modular design is not only a practical concept, but an essential one. The architecture of a modular system can be characterized as "a small set of modules that adhere to some intermodule communication protocol" and are "interconnected by a small set of rules to produce a system that performs the desired algorithm" (Fuller, 1973a). With the future availability of a sufficient variety of LSI modules guaranteed, the system design process could consist of the selection of standardized modules from inventory, and their configuration, by means of an interconnection discipline, into a system that is architecturally suited to the application. The most appropriate module would be chosen to implement each function required (Cooper, 1973). This could, in part, eliminate the complexities and

inefficiencies of programming general purpose computers for applications with characteristics that lend themselves to specialized hardware organizations. The module approach takes advantage of the economic benefits of mass production of standard units with the ability to customize systems.

Modular computer systems are entering their second generation, characterized by an attempt to incorporate the developments of LSI circuits and the computer-on-a-chip into the set of building blocks. This generation will probably create systems that are modular at the PMS level (Ellis, 1973), (Fuller, 1973a). PMS represents the top level of computer structure (Bell, 1970), (Bell, 1971), essentially the information flow level. There are seven basic components, each distinguished by the type of operations it performs: Processors, Memories, Switches, Controllers, Transducers, Data, and Links. The fine structure of information processing is overlooked in order to focus on a small set of components working on a homogeneous medium called information. A succinct notation has been proposed (Bell, 1972) to aid in the analysis of various aspects of computer systems, for example, the effect of the distribution of I/O devices and controllers on I/O rate and throughput, or the prediction of system performance with the addition of specialized hardware. One advantage of the notation is its fluidity - it presents a highly compressed description of a system but allows the selective

amplification of specific aspects that are of particular interest. The structure of the different computer systems discussed later in this thesis will be described using PMS in order to provide a common basis for comparison.

With the wider acceptance of PMS modules as basic system building blocks, it will become economical for thorough design, testing, and documentation to be done by the manufacturer of each module. It is sufficient for the user to verify the macroscopic facilities provided against the specifications for the module (Joseph, 1972). A parallel can be drawn to the testing procedure in an operating system designed and implemented using Dijkstra's approach of a layered, abstract machine. Each level of software presents a set of functions to the layers above it. The next higher layer is only added when the previous has been rigorously tested and has been found to function correctly. Dijkstra (1968) reported that "the only errors that showed up during testing were trivial coding errors..." It was found that the use of hardware modules at the PMS level translated the paper design of systems to an operational system that operated as specified (disregarding wiring errors) (Fuller, 1973a). The disadvantages of PMS modules are that they are more costly and have lower performance than standard logic designs by at least a factor of two. This is the necessary payment to achieve the goals of flexibility, short design time and

expandability (Fuller, 1973a), but one which is being reduced to trivial cost by technological progress.

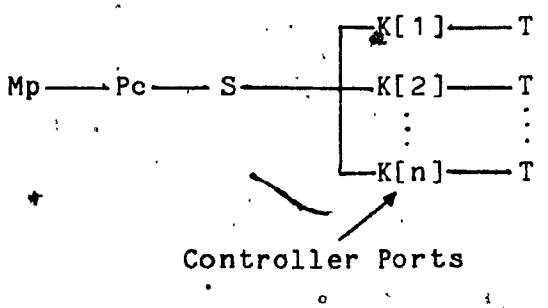
At the present level of commercially available LSI, the designer already has sophisticated hardware modules with which to build systems. Furthermore, these modules are becoming more complex to meet increasing performance requirements and are being spurred on by technological advances. Powerful microprocessors and 16K bit dynamic RAM on a single chip are presently available from a number of manufacturers (IBM has recently announced new products incorporating a 64K memory chip). Microprocessors can be considered as sophisticated control modules. Using them as the basic units of design creates multiple processor distributed systems, with a potential for high reliability and very high throughput.

The two main difficulties with this architectural concept are how to interconnect a number of processors economically and how to program them to cooperate effectively (Fuller, 1973a). Some possibilities for the interconnection of computer modules will be outlined in Section 2.4. The synchronization and communication among these modules will be discussed in Section 2.5 on operating systems. Although modular computer systems have not had much of an impact to date, research in this area will have a major influence on future computer systems and spawn a variety of new architectures and applications (Ellis, 1973),

(Joseph, 1972).

2.3 A COMPUTER MODULE (CM) AS A DESIGN UNIT

The next logical step to more complex modules is to combine PMS elements into a single module. Since it is made up of the same components and has the same structure as a computer, it has been called a Computer Module or CM. Increasing the functional power per module reduces the number and cost of the interconnections. It has been proposed (Bell, 1973) that a CM consist of a processor and memory of about minicomputer complexity (equivalent to today's microprocessor) and ports (controllers) specifically designed to execute concurrently with the processor and handle a variety of communication and line protocols. The structure can be illustrated in PMS as follows:



A typical CM would include one microprocessor with a minimum of 1K bytes memory and two to five I/O ports. The design of the ports is critical because they may be used to interconnect to and communicate with other different CMs.

The CM becomes the basic design unit. A range of CM types will be required in order to be able to provide a range of user and system functions. This can be provided by CMs that differ only in memory size and port design because they are programmable. Each CM will have one or more sequential processes implemented in the module to deliver functions required by the operating system or application.

Since microcomputers correspond so closely to CMs in terms of their architecture, complexity, and capability, the current offerings of microcomputers enable the practical implementation of the concept of a CM as a design unit. A computer system will consist of a set of CMs, that is, a network of microcomputers. The Cm* multi-microprocessor system is one implementation of an interconnected set of computer modules reported in the literature (Swan, 1977). Each CM has been implemented by a DEC LSI-11 microprocessor, standard LSI-11 bus, 64K or 128K bytes of memory, and perhaps, one or more I/O devices. Included is a local switch which connects the CM with the interprocessor communication structure. The CMs are organized as clusters presided over by a communication controller (Kmap), a general-purpose processor with writable control store. This permits an efficient implementation of a variety of communication mechanisms ranging from shared memory to message systems.

The functions provided by each CM must be chosen so that the individual components operate relatively independently to reduce system contention and communication traffic. An approach suggested by a number of researchers (Arden, 1975), (Poujoulat, 1977), (Fancott, 1977), (Browns, 1977) is to identify the functions provided by the operating system and to implement each in a separate microcomputer. These micros are interconnected and the communication method among operating system routines is modified to work in the distributed environment. Protocols for distributed interprocess communication in a network have been proposed to ensure correct and efficient communication (Metcalfe, 1972), (Probst, 1977).

Though designing with such modules can simplify system design, the inflexibility of structure below the module level can result in suboptimal use of resources and lowered performance. Microprocessors, however, have become so inexpensive that using as little as ten percent of the computing power of an existing module is usually cheaper than designing and programming a custom unit that would do the job optimally, that is, with the fewest electronic components (Toong, 1977).

2.4 HARDWARE INTERCONNECTION STRUCTURES/TOPOLOGY

Computer systems can be interconnected in a variety of ways. These can be classified according to two criteria: 1) the looseness or tightness of their connection, and 2) the topology of their connection. Research in the integration of several small processors into one tightly-coupled processing system to produce computing power equivalent to a much larger machine, has been reported. Specifically, the topology and interconnection structure of C.mmp (Wulf, 1972), Pluribus (Heart, 1973) and CMs (Fuller, 1973b) will be described. Much experimental investigation in the connection of independent (and often different) processing systems to form a distributed computer network to share hardware and software resources, has been done and can be found in the literature. The structure of the Distributed Computing System (DCS) (Farber, 1972b), Spider (Fraser, 1975), Aloha (Abramson, 1973) and ARPANET systems (Roberts, 1970), (Heart, 1970), (Kahn, 1972) will be described.

Since there are many possibilities for the physical interconnection of processors and memories, each with its own naming convention, a common taxonomy, proposed by Anderson (1975), will be used to classify and describe them. In this system, the environment is simplified to three archetypes: Processing Elements (PEs- hardware units on

which processes execute), paths (the media by which the transfer of information takes place), and switching elements (the entities between sender and receiver which have enough intelligence to make decisions regarding the destination of a message). The choices for interconnection of the computer system can be viewed as a tree, whose branches represent message transfer strategy, control method, and path structure. The result of tracing a path through the branches results in a system architecture (Figure 2.1).

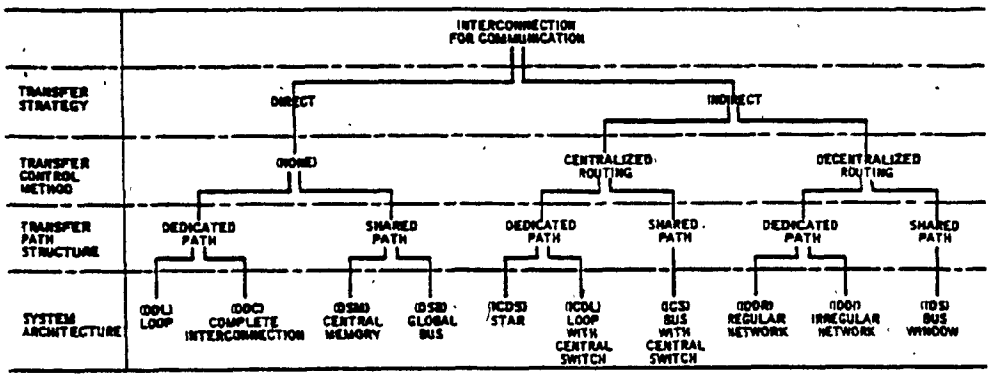


Figure 2.1 Hardware Interconnection Taxonomy

One of the most traditional methods of interconnecting computer systems is the Direct Shared Memory (DSM) or multiprocessor architecture, in which two or more processors communicate by leaving messages for one another in a commonly accessible memory or shared peripheral. An example of such a contemporary multiprocessor is C.mmp (Wulf, 1972), which interconnects a maximum of 16 processors with 16

memory modules through a crossbar switch. The scheduling and coordination of processors is performed by operating system programs that check a shared data base. Two to seven 'locks', depending on the path through the scheduler, protect the data items by guaranteeing mutual exclusion.

In Pluribus, BBN's multiprocessor IMP (Heart, 1973), modules made up of two processors, a small amount of local memory, and bus couplers are interconnected on a global bus in much the same manner as a CM. The initial design has seven processor buses, two memory buses for shared memory and an I/O bus. These buses are interconnected by bus couplers, effectively Indirect Decentralized Shared (IDS) or bus windows, that constitute a distributed crosspoint switch. The use of both local and shared memory in Pluribus contrasts with the homogeneous shared memory in C.mmp and a hierarchical memory organization (as in the CM design).

Each CM module is associated with only one memory. When multiple CMs are interconnected, the entire memory space is accessible to all CMs via IDS structures. A processor in any CM can access the memory module of another CM without the remote processor's cooperation. However, the hierarchy of memories causes an overhead in passing through each level of structure, so frequently used code and data should be stored in local memory. Communication between processes occurs at the single word level and all routing is done by mapping hardware over high performance buses

(Fuller, 1973b). Each of these examples exhibits a tight coupling of processors.

The alternative is to interconnect computers (that is to say tightly-coupled PMS and CM components) in a network or loosely-coupled arrangement. In one such system (Arden, 1975), a number of program processors, each with high speed cache and a request bus, are connected by a single Direct Shared Bus (DSB) to a service centre of four microprocessors that provide memory management, process management, file management and I/O, and monitoring and protection. This group of processors is combined with a very large, hierarchical primary memory to form a cluster. To connect an arbitrary number of clusters together, a ring structure, Direct Dedicated Loop (DDL) has been proposed.

DDL or loop organizations have also been used successfully for the interconnection of geographically dispersed minicomputer systems (Pierce, 1972), (Farmer, 1969), (Manning, 1977), (Farber, 1972b), (Reames, 1975), (Jafari, 1978). The loop consists of a high speed digital communication channel to which PEs are attached through ring interfaces. Messages circulate around the loop in one direction, from source node to destination, with intermediate PEs acting as relay units. This can significantly delay the arrival of a message at its destination which should be considered in choosing a loop structure. Conversely, the ability to broadcast messages to

all nodes with minimal overhead, the low startup cost and ease of adding new nodes is an advantage of the topology. As well, loops can be interconnected via a 'gateway' or 'switching' computer to configure more complex systems and interconnect to larger networks such as ARPANET or DATAPAC. Fault tolerance can be provided by a fully redundant loop to complete a broken line (Hassing, 1973), a bypass switch to disable a malfunctioning processor, and software to discover failures and initiate recovery (Farber, 1972b).

To transfer information, most loops employ the concept of fixed-size frames or 'slots' because this simplifies the message transmission protocol and ring interface (Pierce, 1972), (Farber, 1972b), (Hassing, 1973). A combination of hardware and software ensures that no single node, even with malicious intent, can saturate the entire bandwidth of the loop. The disadvantage of fixed-size frames is that message space on the loop is wasted for short messages, while elaborate disassembly and reassembly techniques using frame size packets are required for long messages. The transmission of variable length message frames has been proposed (Farmer, 1969) but is severely handicapped by the inability to allow simultaneous message transmission. To overcome this, a new message transmission technique was developed which combines the benefits of both Pierce (1972) and Newhall (Farmer, 1969) loops, that is, the capability of sending multiple, variable length messages. This technique

is the basis for the Distributed Loop Computer Network (DLCN) described in papers by Reames (1975) and Liu (1977). In DLCN the loop interface automatically buffers, transmits, acknowledges and controls messages without the need for software supervision or centralized control. As should be expected, DLCN reports improved throughput and shorter response time than both Pierce and Newhall loops. In another implementation of the loop structure, two separate loops have been used, one reserved for data traffic and the other for control messages (Jafari, 1978). Jafari reports an improvement in throughput and response time over the DLCN technique.

A tradeoff in any interconnection scheme is the extent of distribution of the switching function. In contrast to DDL structures, loop systems with a central switch are categorized as Indirect Centralized Dedicated Loops (ICDL). Messages are placed on the loop by the sender, removed for an address mapping operation by the central switching element, then replaced on the loop properly addressed to their intended destination. The service appears as a direct channel linking the sender to the correspondent for the duration of a conversation (Fraser, 1975).

In a system interconnected by a single, common bus, the same decision must be made about control philosophy. Access to the common bus is shared among the PEs by an allocation scheme. Messages are sent directly from the

source PE onto the bus, to be recognized and accepted by the proper destination node(s). This is a Direct Shared Bus (DSB) architecture. Alternatively, a PE can acquire the bus and send the message to a central switch, where it will be readdressed and transmitted to the proper destination. The use of a central switch gives an Indirect Centralized Shared (ICS) architecture.

The Aloha system (Abramson, 1973) is a unique example of an ICS structure since it uses a 24K-baud full duplex UHF radio channel as its bus. A small interface computer acts as the switching element between the PEs, a collection of minicomputers, terminals, RJE devices, and an IBM 360/65. Messages from the different stations are queued on a FIFO basis and transmitted when the channel is available. To handle the bursty nature of messages from remote stations without sharing the bus or complicating the protocol, a random access communication method was adopted. Messages are sent in the form of fixed blocks of 40 or 80 characters plus header information without any central control or synchronization. Extra protection is given to the important identification and control information in the header by assigning 16 cyclic error detecting bits. This allows the switching computer to discard invalid information after receiving only three words of the message. The sender automatically retransmits the message if it does not receive an acknowledgement within a given time period.

Systems which are distributed over large distances and commonly referred to as computer networks usually can be classified as Indirect Decentralized Dedicated Irregular (IDDI) structures. Networks in the IDDI category include store and forward, packet switching networks such as ARPANET (Roberts, 1970), (Heart, 1970), (Kahn, 1972), Cyclades (Pouzin, 1973), and DATAPAC (Mellor, 1977). Networks of this size and complexity are probably less useful as models for the interconnection of microcomputers or the distribution of operating system functions. They have been mentioned for completeness and as a possible source of ideas.

This section has discussed various interconnection schemes currently in use to show the wide range of possibilities. The logical communication path among processes in a system can be mapped onto a physical interconnection of multiple processors (Brinch Hansen, 1971), (Wecker, 1973), (Walden, 1972).

2.5 OPERATING SYSTEM DESIGN

Developments in integrated circuitry and distributed processing are having a great impact on operating systems. Electronic components are able to perform increasingly complex functions, for example, the transmission and acknowledgement of messages or low level communication

protocols (SDLC), that used to be considered within the software domain. Multiprocessor systems and distributed computer networks necessitate the development of operating systems and software for efficient and convenient sharing of resources among many processors. Many distributed operating system designs and implementations have been reported in the literature (Mohan, 1977). This section will describe some of these developments and examine the design goals, logical structure, operating system primitives and innovative features that have been proposed. The main focus will be on the techniques used for communication among processes.

2.5.1 STRUCTURE, SYNCHRONIZATION AND COMMUNICATION

Dijkstra's "THE" Multiprogramming System is the first operating system described since it was one of the earliest developed and has served as a model for many later systems (Dijkstra, 1968). The system was designed to process a continuous flow of user programs. The multiprogramming system, made up of user programs, operating system routines, and peripheral control programs, is organized as a society of sequential processes. The harmonious cooperation of processes is regulated by means of two explicit synchronization primitives provided by the operating system (at the same level at which processes exist). These two primitives, P and V, operate on a specialized integer data type called a semaphore.

Semaphores can be used to coordinate the access to a shared address space by two processes (by ensuring that processes only access the space within a critical section delimited by P and V), to control a producer/consumer relationship, and to implement the mutual exclusion required for resource allocation. On a single processor system, semaphores can be implemented by masking all interrupts for the duration of the P and V operations (to guarantee their indivisibility). On a multiprocessor, special hardware is needed to guarantee that only one processor at a time can update the semaphore. A memory interock or TEST AND SET instruction as provided on IBM S/370 systems ensures exclusive access to a memory location. In the VENUS system (Liskov, 1972) P and V are machine instructions, implemented in microcode on a minicomputer.

A further extension to semaphores has been implemented on the Honeywell Series 60/Level 64 (Atkinson, 1974) and by an operating system for a network (Akkoyunlu, 1972). A short fixed-length message area is associated with each V operation, which the system transmits to the process executing the corresponding P operation. This is useful to describe the event; for example, status information on I/O.

Two higher level constructs, with the advantage that they combine the operations and data structure in a single entity, have also been proposed. Secretaries associate the procedures with the data structure on which they operate and

implement the two together in a single process. The monitor concept, attributed to Brinch Hansen (1973) and Hoare (1974), combines the shared variables, the set of meaningful operations on them, and the mutual exclusion in one construct. This enables monitors to be defined in the syntax of a higher level language and checked at compile time, which simplifies their use and decreases the probability of error.

The structure of the "THE" system displays a strict hierarchy of five levels. At level 0, the system implements processes, which define the first level of abstraction. This includes the logic for processor allocation to a process, dependent on a priority scheme to achieve quick response of the system where this is needed and time slicing to prevent any process from monopolizing the system. At level 1, all information is organized in terms of segments. At level 2, the message interpreter process handles the allocation of the console keyboard to the process addressed by the operator. At level 3, a process handles the buffering of I/O to logical communication units supported by a limited number of actual peripherals. At level 4 are user programs and at level 5 is the operator. In this way, a layered, abstract machine is created. The concepts of semaphores for synchronization and a layered, abstract machine implemented as a hierarchical system of sequential processes have had a significant impact on operating system

design.

The operating system for a network environment developed at State University of New York (Akkoyunlu, 1972) is based on the concepts developed by Dijkstra as outlined above, that is, synchronization via semaphores and a layered abstract machine approach to software development. The key requirement for an operating system that must be expandable to a network is the uniform treatment of local and remote transactions. Communication is provided through the mechanism of ports, an abstract structure by which a process can communicate with external objects (files, devices, or other processes) uniformly. The port concept is presented in detail in papers by Balzer (1971) and Walden (1972). In order for a transaction to take place through a data port, both parties involved must indicate a willingness to transfer information by issuing matching requests. The owner of the port makes a request, with the option of specifying a particular process with whom he wishes to communicate. Each new request is checked against a queue of previous requests that are currently waiting. On a match, the queued request is serviced and removed from the queue.

The system itself is structured as a hierarchy of levels. At level 0, the nucleus interfaces to the hardware and provides CPU multiplexing. Semaphores and their P and V operations are implemented at this level. At the LE (Logical Element) level, processes are defined. Above this

level is the DP (Data Port) level which provides the IPC (Interprocess Communication) facility for processes. The KI (Known Item) level ensures the orderly access to the data ports and ensures the integrity of the system. The user level resides above these four levels.

In the MULTICS system (Spier, 1969), a completely generalized, modular unit called the traffic controller, has been provided as part of the control supervisor program to assist the programmer to coordinate the activities of two or more processes. This interprocess communication facility is used extensively by the system itself. The exchange of data among cooperating processes takes place in a shared data base, that is a data base with read/write access by all communicating processes. To control multiple access to the data base, there is a need for a locking mechanism to ensure exclusion among interfering processes. This is provided as lock and unlock functions based on the hardware test and set instruction. As well, block and wakeup functions are provided by the traffic controller, to enable processes to share the central processor and order their execution.

The data is organized in the form of messages. Communication is achieved by an exchange of these messages in a commonly accessible mailbox whose identity is known to each process by common convention. This implies an IPC setup to enable a process to gain knowledge of a mailbox, to agree on an 'empty' state, and a mechanism to enable a

process to set the state of the mailbox to not empty which another process will interpret as a message having arrived.

The StarOs operating system, which was designed to execute on a system of interconnected CM modules, the Cm* multi-microprocessor, uses the same type of mechanisms as MULTICS: an IPC implemented via messages deliverable to a mailbox and explicit controls to perform a block to await an event (Jones, 1979). StarOs implements a mailbox type object capable of buffering messages and a region queue to store the invoker's name and an event. Two functions, send and receive, are provided. Send will deliver a message to a registered receiver on the queue or buffer the message and deposit it in the mailbox. When the message has been transmitted, the receiving process is signalled that the particular event has occurred. If there was no process waiting, the message is deposited in the mailbox. Later, when the receive function is is invoked, it will return a buffered message from the mailbox. The mailbox functions are implemented partially in software and partially in microcode.

In the RC4000 Multiprogramming System (Brinch Hansen, 1971), as in Dijkstra's view, the environment is seen to consist of concurrent, cooperating processes that require the ability to synchronize their activity. A system nucleus implements a set of primitives that enables processes to communicate (by sending messages or answers) and synchronize

(by waiting for messages or answers). The buffer space for messages comes from a common pool administered by the nucleus. The send function copies a message into the first available buffer within the pool and delivers it to the queue associated with the named receiver. The receiver is activated if it was waiting for a message. The sender is allowed to continue execution. Rather than always waiting for the next message, two additional primitives are provided that enable a process to await the arrival of any message or answer and to serve its queue in any order.

The nucleus is also responsible for the creation of processes. This enables a small set of processes to be created that define the basic functions of an operating system and then allows the user to dynamically add new operating systems as descendants of the nucleus, specifically suited, for example, to time-sharing or real-time processing. In the RC4000 system, each function, whether it is arithmetic/logical or input/output, is handled by a process. Each process is identified by a unique name, enabling others to refer to it without knowing its type or actual location in storage. Processes are arranged in a hierarchy, in which the nucleus and a basic operating system are parent processes that create and control new processes (which in turn may act as an operating system for their children).

The implementation of the send function as an explicit data exchange between processes, as opposed to the passing of pointers to the data, is an important concept. It enables the same mechanism to be expanded to a distributed system, where processes may reside in, and in fact, migrate to different processors that are interconnected by a physical path (Metcalfe, 1972), (Walden, 1972), (Wecker, 1973).

The Unix system is based on much the same concept as Brinch Hansen's RC4000 system. Unix is made up of a set of processes organized in a hierarchical manner (Ritchie, 1974). A new process can come into existence only by use of the 'fork' system call. The calling process is split into two independently executing processes that share any files opened by the caller. One of the processes becomes the parent, the other the child. For processes to communicate they must be connected by an IP channel called a pipe that was set up by a common ancestor. The two related processes issue read and write operations, in the same way as they would do I/O to a file. A process that issues a read to a pipe is suspended until another process writes to the same pipe.

Unix has been modified by the addition of a Network Interface Program (NIP) to connect to the ARPANET or other networks, to act as a gateway between networks, or to configure a multiprocessing system made up of several Unix

systems (Chesson, 1975). The NIP consists of a Network Control Program (NCP), protocol programs, and network special files. The NCP is made up of a kernel that is part of the Unix system and a 'daemon' which runs as a continuous background user-level process.

Medusa is a multi-user operating system under development which, like StarOs, was designed for Cm* (Ousterhout, 1980). The aim was to understand the effect of the distributed Cm* hardware on operating system structure, and consequently, Medusa attempts to capitalize on and reflects the underlying architecture. This gives rise to two important software issues - partitioning and communication.

In traditional multiprocessor systems, the operating system is implemented in one processor's memory, with other processors executing the code remotely. This is not feasible in Cm* because of the high overhead of accessing remote memory and, in any case, is too fragile in terms of reliability. At the other extreme, all operating system code could be replicated in each processor (similar to the approach taken in networks) but the small size of local memory discourages this approach. Instead, the operating system was divided into disjoint utilities using the decomposition scheme suggested by Parnas (1972) and distributed among the set of CMs. A given processor is permitted to execute the code for a particular utility only

if it can do so locally. Otherwise, it must invoke the remote utility by sending it a message. The boundaries between utilities are rigidly enforced (for protection) and are only crossed by messages. Messages are transferred via pipes (similar to those of Unix) so as not to restrict either the location or organization of the senders and receivers.

Roscoe is a distributed operating system designed for a network of microprocessors (Solomon, 1975). Currently the system consists of five DEC LSI-11s with 56K bytes memory and word parallel lines to one or more other LSI-11 machines (and a PDP-11/40 running Unix for loading and development work). The software is written in the C programming language (except for a small amount of assembler).

Two processes are connected by a link - a one-way logical path (the link structure was first proposed by Baskett (1977) for the Cray-1 uniprocessor). The holder of the link may send messages over the link to the owner who receives them. The owner creates the link and never changes. The owner may specify the properties of the link, for example, that it may not be copied or that he should be notified if the link is destroyed. The chief function of the kernel, implemented in each machine, is to support links and messages by providing service calls to create and destroy links and send and receive messages. As much as possible, operating system functions are provided, not by

the kernel, but by processes. Each kernel maintains a pool of buffers that can be allocated for incoming messages and to local processes for outgoing messages according to a simple strategy. Three priorities of request are possible: high (satisfied if any buffers are available), medium (satisfied if a quarter of the pool is available), and low (satisfied if half the pool is available). Another alternative suggested is to implement a buffer quota for each process.

In IBM's DPPX operating system, the functions and data are dispersed in a network, with coordination performed by one logical manager. The system supports connection with peer systems (other 8100 systems), hierarchically defined hosts (System/370, 303x, and 4300 processors), and terminals. All system objects, for example, programs, configuration definitions, display maps, and data set (file) definitions have been implemented as data sets. A single command language is provided for user access to all system functions.

The DPPX system is constructed as a hierarchically defined set of layers. The Control Program is made up of three parts: a base layer that creates and supports the tasking structure and supplies synchronization services that are implemented through queues and locks, a layer that provides extended supervisor functions to support management of processor storage, program contents, timer services and

error handling, and a layer that provides resource management functions such as the allocation of data sets. I/O Services are consistent with the functional layering of the SNA architecture (for a thorough description of SNA, read the textbook by Cypser, 1978).

2.5.2 NETWORK OPERATING SYSTEMS

Basically, there are two strategies for the implementation of network operating systems (Forsdick, 1978). The fundamental operations are integrated with the functions that make up the operating system and are implemented directly on the hardware, for example, DCS (Farber, 1972a). Alternatively, the basic functions provided by the host operating systems are used as the building blocks with interhost communication and a distributed file system added, for example, Reames' and Liu's DLCN.

The Distributed Computing System (DCS) is a facility made up of a number of different processors connected by a digital communication loop. Control of the loop and responsibility for resource allocation and scheduling are distributed and shared among all processors to gain resiliency. Each processor interfaces to the loop via a special piece of hardware called a ring interface which consists basically of buffers, a shift register, and an

associative memory. This memory contains the names of all processes active on the attached processor, enabling messages to be sent addressed by process name. Since processes communicate over the loop using messages, the same mechanism was extended to interprocess communication within a processor. A process wishing to communicate with another, merely transmits a message to it. If the destination process is not active on the same processor, software will route the message to the loop. Otherwise, the message would be transferred directly to the destination process. This scheme resembles the IPC proposed by Brinch Hansen (1970) and consequently, also provides a method of synchronization. Since no common memory is required, modules of the operating system can be distributed among the processors.

The operating system exhibits a hierarchical, modular structure (similar to Dijkstra's "THE"). Level 0 is basically a multi-priority, round robin scheduler. Level 1 is the software that implements the IPC. Each process has a set of message queues called channels, one of which is designated as the output channel. All others are input channels, associated with an originating process name or class from which messages will be accepted. When a process wishes to send a message, it places it on the output channel and calls the level-0 scheduler to activate the communication software. When a process wishes to wait for a message, it signals the scheduler which marks the input

channel on which to wait and makes the process inactive. When a message is delivered to a wait channel, the communication software returns the process to the active state. Level 2 checks the loop and processors for malfunctions. Level 3 consists of the resource allocation routines and level 4 is the user and service processes.

The Distributed Loop Computer Network (DLCN) provides a unified system consisting of small and medium scale computers, terminals, and peripherals connected as a geographically local network by a communications loop (Liu, 1977). Functionally, users are not even aware of the system's actual organization. One of DLCN'S major objectives was to design an appropriate message communication protocol for the network which would also simplify the implementation of distributed, low-level primitives for network operating system functions. The communication protocol (called DLMCP) for message transmission on the loop is a bit-oriented protocol (like IBM's SDLC) with a distributed control discipline. Messages are addressed by process name which is mapped into a two component physical address: a 7-bit loop interface address for locating the processor and a 5-bit number which identifies the process. Four types of messages are provided for network functions. Information messages transfer text between communicating processes. Acknowledgement messages are automatically generated by the interface hardware as

required, either for each message received or for a block of messages, as specified by the sender. Control messages implement privileged low-level primitives for accomplishing some of the basic functions of the network operating system (DLOS), for example, locate a process, call a remote program, or synchronize. Diagnostic messages are used for error detection and recovery during normal operation and for system initialization and loading at start-up time.

All processors execute components of the network operating system, DLOS, whose primary role is the conversion between local and network-wide representations of information. Therefore, it can be implemented differently on each host, as required to interface the local operating system to the network.

2.5.3 OPERATING SYSTEMS TO FUNCTION MODULES

An operating system can be viewed as a set of parallel, cooperating processes, where the system is defined in terms of the supervisor and control functions provided and their interrelationship (Coffman, 1973). All processes are assumed to execute asynchronously, coordinating their activities through information transfer. They exhibit the same properties as interconnected hardware devices: independence except for activation and communication (Wecker, 1973), (Habermann, 1972). This suggests that a

mapping of processes onto hardware modules would be feasible.

There is considerable motivation for the development of structured, modular operating system functions which can be implemented in low cost LSI hardware components. The overhead in current operating systems threatens to defeat their very purpose - the efficient use of hardware resources. It was found that over 50 percent of the instructions are executed for system support and control (Joseph, 1976) and that the code required 32K to 128K of 32-bit words storage (Habermann, 1972). A computer overloaded with highly complicated software will not stabilize very well, and consequently, will be unreliable (Hopper, 1976).

The lack of a uniform communication philosophy tends to complicate the operating system and can cause problems of synchronization, scheduling, and reliability. A solution to this problem is to establish a common, system-wide communication facility, based on information exchange. The system displays homogeneous communication (all processes use the same protocol) and location transparency (the protocol is the same for both local and remote destinations). This allows system processes to migrate within the configuration, from one processor to another or from software to hardware. For example, device controllers, with added intelligence, could respond to messages as opposed to I/O instructions and return messages as opposed to interrupts to signal programs.

III. MMCMS EXPERIMENTAL PROTOTYPE

This section describes the environment, development work, and implementation of an experimental system used as the basis for the Multi MicroComputer Module System (abbreviated as MMCMS). The goal of this work is to demonstrate the functioning of the software interface and message-based communication, and to investigate the choices when offloading a function from a single processor operating system to a distributed architecture. The message transfer facility could support the distribution of an operating system organized as cooperating processes that communicate and synchronize via an exchange of messages. It should be noted, however, that the scope of this experiment is limited to a single user, single task operating system. The setup was designed to be a flexible, modifiable test system that would implement one physical mapping of the logical communication concept. Its choice was determined by available hardware and software, and is not intended to be a demonstration of an optimal design.

3.1 DESCRIPTION OF THE TI ENVIRONMENT

The architecture, instruction set, and operating system of the TI are described with explanations and diagrams of the communication and control flow. The

description of architectures is supplemented by Bell and Newell's PMS notation (Bell, 1971). The design and implementation of a logical communication structure among interconnected computer modules, including a distributed communication routine, protocol, and message format is explained in detail.

3.1.1 ARCHITECTURE OF THE TI980

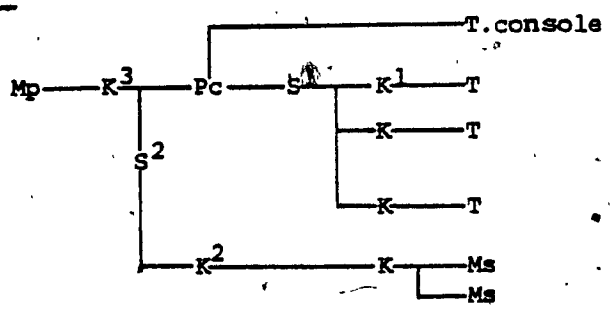
The Texas Instrument 980 Model B minicomputer is typical of its class of machine. The TI980B is a single-address (plus index register), general purpose 16-bit digital computer with integrated circuitry designed in the 1960's. It has a status register (for processor and hardware conditions) and eight addressable registers, not truly general purpose as not all registers can be used for all instructions. Arithmetic operations are performed in 2's complement. There are 99 basic instructions of one to three words in length.

The system includes memory parity, hardware program relocation, one auxiliary port, one Direct Memory Access (DMA) port, and four I/O bus ports with interrupt. Peripheral devices are connected to one of two separate buses depending on the speed and length of transfer. For slow, character oriented devices, the central processor transfers a single word (8 bits control and 8 bits data) via

the I/O bus to or from a register or memory location by issuing an RDS or WDS instruction. Using program controlled I/O, transferring a line or block of data involves looping through a program sequence that checks that the device is ready and issues the RDS or WDS for each character. Alternatively, the I/O ports can be polled by reading status from slot 13 to find out which ones have interrupted. For fast, block transfer devices, the central processor has only to initiate the data transfer between the device controller and memory by issuing an activate instruction. The DMA port handles the logistics of the transfer, requesting access to the memory from the memory controller on a cycle stealing basis with the central processor.

The PMS representation of the TI architecture (Figure 3.1) shows how it is an outgrowth of the classical configuration for a Computer/C:=Mp Pc K T through the use of two distinct information paths, the I/O bus (S and links) and the DMA channel (S with links) and the addition of the memory controller (K).

The architecture of the TI980 is characterized by the use of a powerful memory controller which performs all addressing decoding, parity generation and checking, timing, and control functions on the memory. With these capabilities, the memory and memory controller comprise a relatively independent processing unit whose function is to accept addresses and deliver or store information on



S^1('I/O Bus; duplex; iu: 1 w)
K^1('peripheral controller and interface)
S^2('DMA Channel; duplex; iu: 1 w)
K^2('Block Transfer Controller/BTC)
K^3('Memory Controller; i: Pc|K('BTC))
T(paper tape, card reader, VDU console, line printer, plotter)

Figure 3.1 TI980B Architecture in PMS Notation

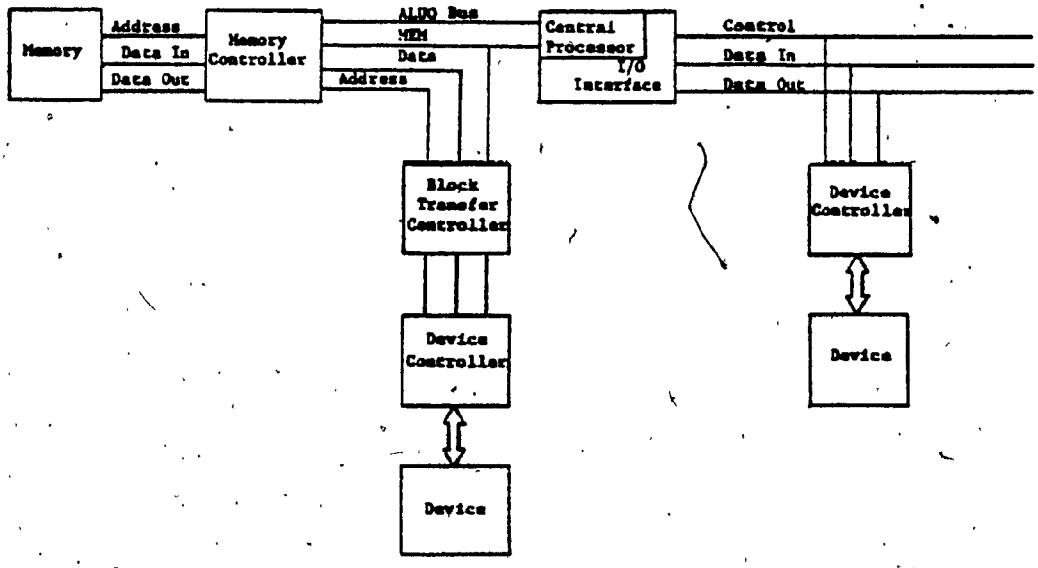


Figure 3.2 TI980B System Block Diagram

request. It acts as a switch between the processor and the DMA port, but can accept requests from both at the same time. This organization allows considerable flexibility of operation and ease of interfacing with any device which requires direct access to the memory. This is shown clearly in the block diagram of Figure 3.2.

3.1.2 OPERATING SYSTEM OF THE TI980

The standard operating system, written in assembly language, creates a single program execution environment. The configuration available consists of a VDU with keyboard, card reader, line printer, and moving head disc with removable cartridge. The user is able to communicate via an operator's console to request the operating system to perform any of a given set of functions. These can be grouped into four subsystems:

1) a system loader,

2) a program/machine housekeeping system to monitor and act on power fail, memory protect violations, internal interrupts, etc.,

3) an I/O system that connects logical units to physical devices and performs all I/O operations, and

4) a disc file management package which builds, identifies, and maintains files on one or more disc volumes.

In general, functions provided in software and accessible to the operator can be implemented at one of two distinct levels:

1) as a command available through the operator communication package, part of the operating system

2) as a system program (the object resides on the disc) which is loaded and executed to perform the function and which, in fact, may call the operating system to do privileged operations.

With the distribution of operating system functions to physically interconnected computer modules, a new level of implementation becomes available. A separate module can provide the system function required. For example, the functions of the console device service routine (DSR) can be provided by a microcomputer interfaced to the console device. The micro will be interconnected to the TI minicomputer and communicate with the rest of the operating system by transferring information on the link.

3.1.3 FUNCTIONS OF THE CONSOLE DSR

The first step in the distribution of a DSR to a separate computer module is to be able to replicate the functions that it performs. This section will describe the functions of the console DSR and its interrelationship with the operating sytem (Figure 3.3 shows the system tables

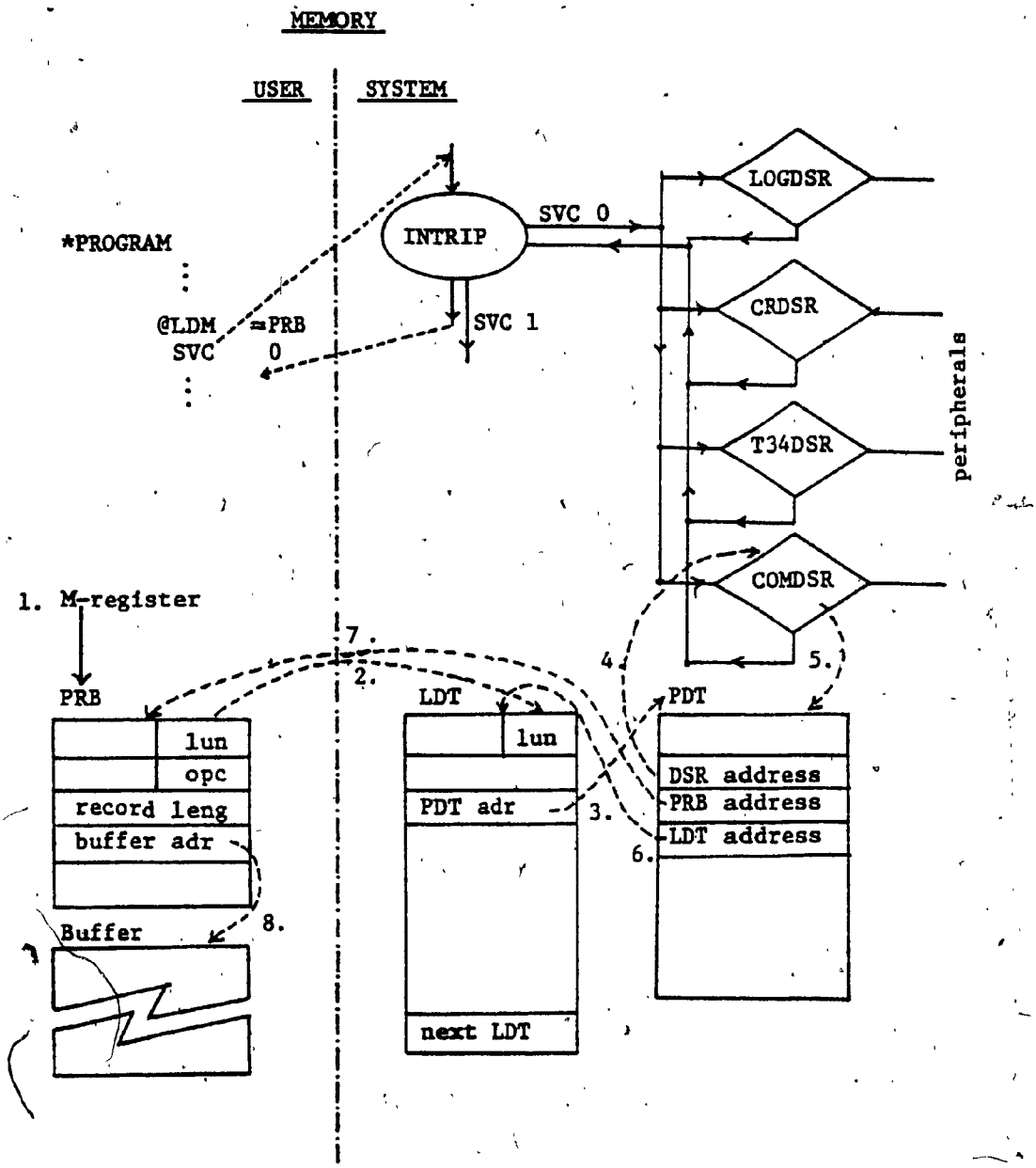


Figure 3.3 Interrelationship among Calling Program, INTRIP, DSRs, and System Tables

accessed by the DSR).

On entry to a DSR, the M register points to the Physical Device Table (PDT) for the peripheral, enabling the DSR to gain access to the Physical Record Block (PRB) (through PDT(2)) and Logical Device Table (LDT) (through PDT(3)). A DSR is responsible for performing a defined set of operations, such as read and write ASCII, or open and close.

On an open, the record length is checked and set in LDT(3) and the device flagged as open, bit 3 of LDT(0), and pagemode (only one screen full of characters will be displayed) is set. Open-rewind sets the ignore bit, bit 3 of PRB(0), and does an open.

On a read, if the device has not been opened, an error message is output to the screen. Otherwise, the open length is retrieved from LDT(3) and the prompt characters are output. The program then loops, reading one character at a time as it is keyed in, and checking for agreed upon special function keys:

ESC - the entire line keyed in is deleted both on the screen and in the caller's buffer

ETX - pagemode is reset

STX - pagemode is set, variables initialized

EOT - end-of-file is sensed

All other characters are echoed - the following also having

a special function:

CR - (carriage) return signifies the end of line and causes a linefeed (LF) character to be output to the screen

BKSP - control H signifies that the previous character is to be deleted both on the screen and in the caller's buffer

All other characters are packed into the caller's buffer. When the buffer is full, the next character keyed in, if not one of the above, triggers a second prompt- a ↓ appears on the screen instead of the character and the cursor backs up to blink under it: ↓ . At this point only the special function characters listed above will be accepted.

On a write, the record length is checked (length>0) before continuing. Then the message text to be output is fetched one word at a time, unpacked character by character and output to the screen. If pagemode is set, a count is kept of the number of lines that have been displayed and output stops when the screen has been filled. At this point, only the following special keys are accepted:

ESC - terminates the output, sets the ignore bit in PRB(0)

LF - continues the output, resets line count

ETX - resets pagemode, continues outputting

On a close, bit 3 of LDT(0) is reset and control returned to the calling routine. Close-eof sets the eof

bit, bit 2 of PRB(0) and then does a close.

3.1.4 COMMUNICATION AND CONTROL FLOW

In the TI operating system, information is transferred between programs by passing a pointer to the appropriate system table or user data block in memory. Control is transferred at the hardware level, via one of the branch instructions. The following is a description of what happens after the TI operating system has been loaded from disc. The program control flow is illustrated in Figures 3.4 and 3.5.

On startup the TI operating system branches to JCSTRT, the entry point of the job control statement processor, JCMAIN. JCSTRT blanks its input buffer and then branches to JCREAD to input a job control record from the console.

For the operating system to perform I/O, a set of parameters must be passed to the DSR of the device. The caller does this by building a five word Physical Record Block (PRB) which contains the logical unit number, opcode, record length, and starting address (refer back to Figure 3.4 for the layout of a PRB). The operating system is signalled and control passed to it by issuing a Supervisor Call (SVC). The address of the PRB is passed in the M register.

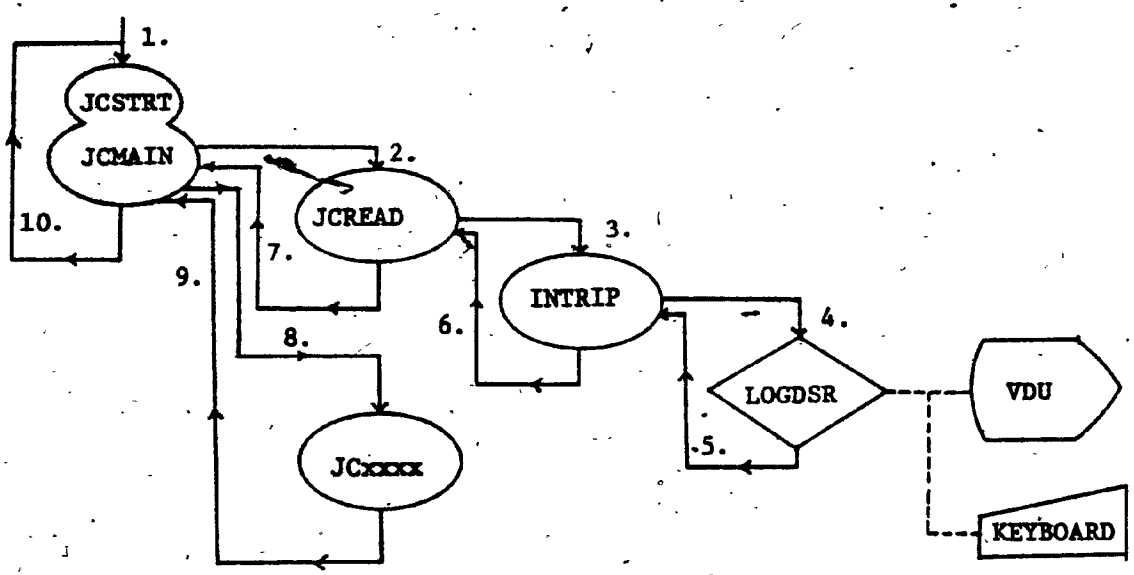


Figure 3.4 Control Flow for Function at the Operating System Level

The SVC is trapped by INTRIP, the internal interrupt handler, and recognized an an I/O request. INTRIP then checks the logical unit requested by PRB(0) against those assigned in the chained list of LDTs. If a match is found, the PDT is linked to the LDT and PRB, and control passed to the appropriate DSR via PDT(1). The DSR, in this case LOGDSR, gets the PRB address from PDT(2), picks up the opcode, and performs its function. Since it is a read, it checks that the device has been opened, gets the record length from LDT(3), and outputs the prompt characters. It then loops, waiting for a character to be keyed in.

The operator can now enter his command. As the first example, a function provided at the operating system level will be requested. The program control flow is illustrated in Figure 3.4. The commands and notation used in the following dialogue are explained in Appendix A.

LU[NOS]) (lists logical units currently assigned)

As each character is entered, it is processed by the DSR, that is, it is checked against a list of control characters and if a match is found, handled specially. If not, it is accepted as a regular character, echoed to the screen (since the console operates in full duplex mode), and packed directly in the caller's buffer. It is at the DSR level that keying errors can be corrected, that is until the return key is entered. On end-of-line the DSR sets the status byte and returns to INTRIP which passes the status to

the caller and returns. Control is given to JCREAD which does some more processing and returns to JCMAIN. The buffer is scanned to 'crack' the fields of the job control message and builds the Field Scan Table (FST). The first entry of the FST, which is the command keyword entered, is compared to a linked list of command names and programs, with control passed to the program with the matching name. In our first example this would be JCLUNS. If no match is found, the convention is that the name is the filename of an object program on disc that should be executed. Consequently a dummy EXECUT(E) command is built. After the function corresponding to the command has been performed, control is passed back to JCSTRT for the entire process to be repeated. In the first example, the system would respond with

```
      2    LOG      3    NULL
>
```

The system is now ready to accept another command. As the second example, a function implemented as a system utility will be requested. The program flow is illustrated in Figure 3.5.

As the second example, a function provided by a system utility will be requested. COPYAL is a routine that copies ASCII records from logical unit 28 to logical unit 27. To list a card deck on the line printer, logical unit (LUN) 28 has to be assigned to the card reader, mnemonic CR. To execute the program, we need only type its name.

```
A[SSIGN],28,CR
```

[EXECUT,][DO,]COPYAL↵

Because LUN 27 has not been assigned the system responds with an error message.

```
LUN 27 ERR 0010 AT xxxx LL zzzz
RETRY>>
```

No recovery is possible so enter N↵ . Assign LUN 27 to the line printer, mnemonic LP and execute the program again.

```
A,27,LP↵
EX↵
LUN 28 ERR 2000 AT yyyy LL zzzz
RETRY>>
```

The card reader is not ready. Press start and wait for the green ready light. Answer Y↵ to the retry message and the program resumes.

If the hopper runs out of cards, the system loops, waiting for more. When an end-of-file card is read, the DSR recognizes it and sets eof, bit 2 of PRB(0), the status word of the PRB. The program checks the status and terminates by returning control to the system, SVC 1. The message COPY COMPLETE appears on the screen and control is passed back to JCSTRT, the idle loop of the operating system.

3.2 DESCRIPTION OF THE SMS300

3.2.1 ARCHITECTURE OF THE SMS300

The SMS300 was chosen as the test module for the implementation of the LOG device service routine. Its architecture and instruction set will be described in the

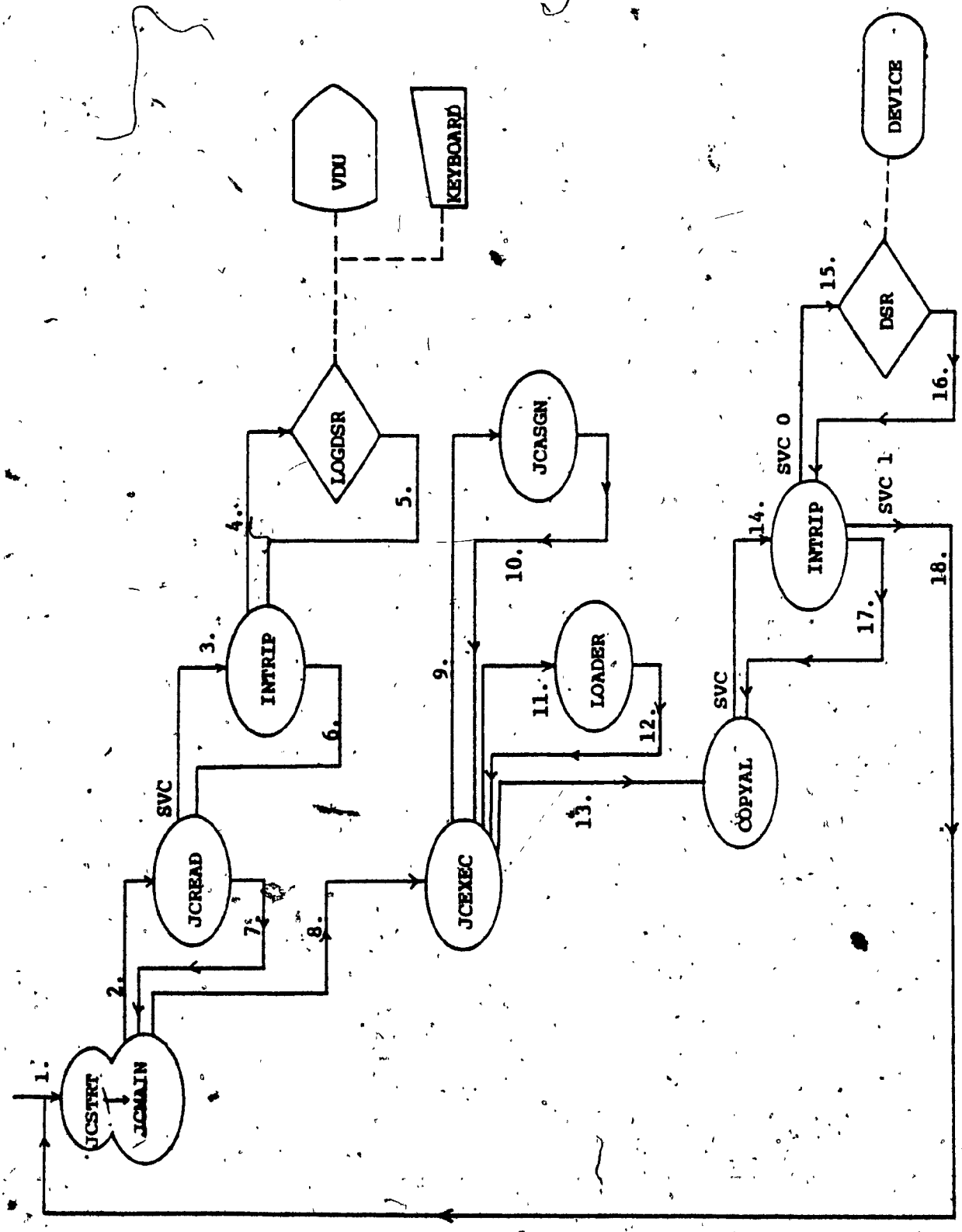


Figure 3.5 Control Flow for Function Provided as a Utility

following sections.

The SMS300 is a small (800 to 1000 gates), bipolar LSI microcomputer that was designed specifically for control applications. There are eight 8-bit general registers, one 1-bit overflow register, and 256 bytes of working storage that can be viewed as an extension of the registers and used for the intermediate storage of variables and I/O data. Two 8-bit address registers (called IVR and IVL) are provided, one provides access to working storage and the other to the IV bytes. The IV bytes are 224 (expandable to 2040) individually addressed I/O connection points. SMS programs reside in a separate memory (16 bits per instruction, addressable to 4K of ROM or PROM), called Program Storage (PS). The components making up the SMS300 and their interconnections are shown in Figure 3.6 . The structure is also displayed in PMS notation (Figure 3.7) for comparison with the TI architecture (Figure 3.1) and structure of a CM (refer back to Section 2.3).

Since the SMS300 is a controller, its bus structure facilitates the connection of peripheral devices. The I/O path is called the Interface Vector and provides a program addressable, buffered, bidirectional path. The user sees it as one or more individually controllable 8-bit registers called the IV bytes. This simplifies the connection of peripherals since the interface does not require logic for recognizing its own address. Although there is no interrupt

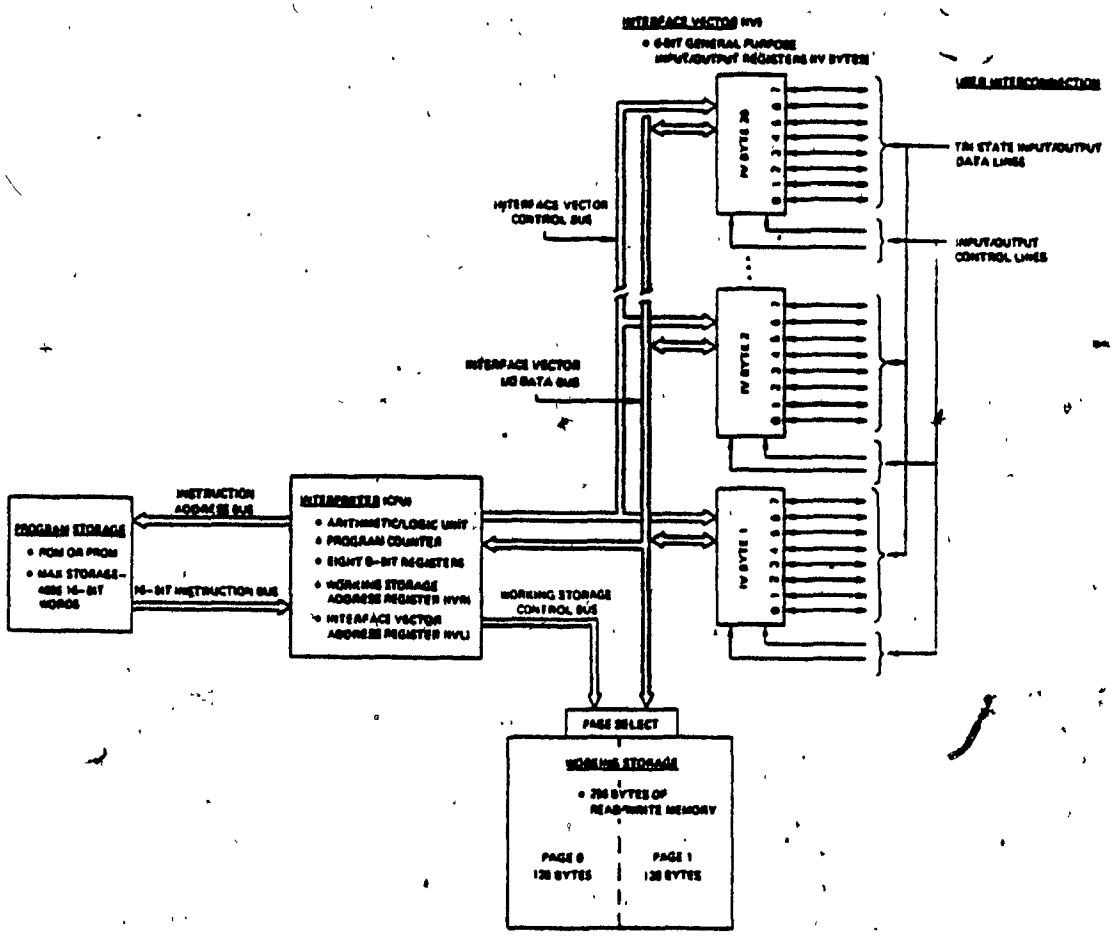
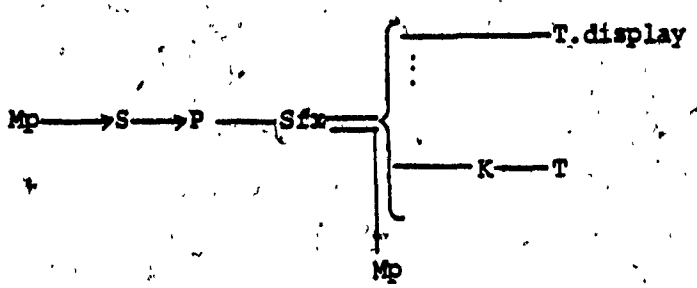


Figure 3.6 SMS300 System Block Diagram



Mp ('Program Storage; 1K 16b; 75 nsec.)
Mp ('Working Storage; 256 bytes)
S ('Interface Vector; #1:28 bytes)

Figure 3.7 SMS Architecture in PMS Notation

facility in the SMS, the architecture is particularly well-suited to a handshaking protocol. This could include the SMS raising the interrupt line of another processor.

3.2.2 INSTRUCTION SET OF THE SMS300

The SMS has an instruction set that consists of eight instructions: 3 arithmetic logical, 2 transfer of data, and 3 program branching instructions. The central processor performs 8-bit, 2's complement arithmetic. The architecture expands the functionality of the instruction set. For example, the MOVE instruction performs performs a load, a store, register move, read byte, write byte, and right rotate. The complete list of SMS instructions and a brief explanation of their operation can be found in Appendix A. Moreover, accessing data is possible on any subset of one byte, meaning that any bit or group of bits is directly accessible in one instruction cycle. Consequently, the information can be compacted for storage efficiency and yet be accessed without lengthy decoding. Any of the instructions can force I/O simply by using the IVL register as one of the operands. This is possible because the architecture is structured such that data, whether at the I/O interface (IV bytes) or in a register or working storage, are handled the same at both the hardware and software levels. This implies that data from an external device can be processed by any of the instructions

immediately, without first moving them to internal storage. In summary, a small, basic instruction set with powerful bit and byte handling capability was implemented in the SMS, appropriate to the intended function of the module, that is, control applications.

3.2.3 INSTRUCTION SET AND PROGRAMMING

This section examines the suitably of the SMS instruction set for the implementation of operating system functions and indicates areas for improvement. An advantage of the SMS is its speed - 300 nanosecond instruction cycle (10 to 50 times faster than most microprocessors). At this speed, operations consisting of three to four SMS instructions still execute at minicomputer speeds. To help clarify programs, the SMS macro facility can be used to build new instructions consisting of more than one SMS instruction. Two approaches are possible for the implementation: emulation of the TI instruction set using the SMS macro facility, or use of a a higher level language suited to operating system design. In the first case, no rewriting is required and testing and debugging can be done on the TI, where hardware and software development tools are available. Afterwards, the communication routine and DSR (in TI assembler) can be assembled for the SMS, using the macro capability to produce correct SMS machine code. This can then be downloaded to SMS program storage and executed.

The advantage of this approach is that the new SMS software is expressed in the same language as the TI software it emulates. It was found that using the macros produced inefficient code, in terms of both speed and memory. The SMS is able to address up to 4K words but the experimental system had only 1K available. As well, the radical differences in the two architectures, specifically addressing modes and the way registers and memory are referenced on the TI, made the macro implementation unattractive. If a PDP-11 minicomputer had been used, operating system functions could have been directly transferred to the LSI-11 microprocessor since it has the same architecture and instruction set as the PDP (Titelbaum, 1975).

The second approach is to build a high-level language that could be used to implement operating systems using the macro capability of the SMS cross-assembler. Two languages that were considered were PL/M, a subset of PL/1 and currently available on some microprocessors (Kildall, 1974), and C, developed at Bell Labs and being used at the University of Waterloo for network development work (Manning, 1976). However, this was a large undertaking not directly related to the specific aims of this research and due to limitations in time and manpower, it never progressed beyond the investigative stage. The implementation of a system language appropriate to building operating systems

and, at the same time, usable on a microcomputer would provide an integrated and powerful tool for the distribution of operating system functions to computer modules.

In the end, the decision was made to build additional multiword instructions using macros to clarify and simplify programming the SMS. As a compromise, only macros with small bodies and minimal overhead were implemented so as to use program storage efficiently. In coding the LOGDSR, the approach was to translate the TI assembler code, on an instruction by instruction basis, taking into account blocks of code that could be optimized using capabilities unique to the SMS.

In general, a shortcoming of the SMS is its use of memory. The architecture requires that all operations be performed through the AUX register and that working storage or interface vectors be selected before they can be accessed. This often requires a move instruction followed by the operation, which uses two words of memory. Also, operations usually provided in an instruction set have to be built up of several SMS instructions, requiring multiple memory locations. These criticisms are mentioned because the memory required to implement a function may become excessive. The DSR function, coded in SMS assembler, requires approximately 400 words of memory. This is very close in size to the TI code, which seems to suggest that savings made using SMS' architecture (better suited to the

application) were lost in a too small instruction set.

3.3 COMMUNICATION IN THE EXPERIMENTAL SYSTEM

Programs in the current TI operating system use a number of different structures for setting up communication. The job control monitor, JCMAIN, passes parameters via a Field Scan Table (FST) to the program providing the function requested. Programs requesting I/O pass control information and a pointer to data in a Physical Record Block (PRB) to the DSR via the M register. When the operation has been completed, the DSR sets status in the PRB and other system tables. Programs requesting the services of the system message writer, MSGSM, format the information (number of words followed by the actual text) and pass a pointer to it in the X register.

3.3.1 COMMUNICATION PROTOCOL

In the experimental system, a program communicates with another program which may reside in a separate function module by explicitly transferring information organized as messages. Eventually, all programs in the system would be modified to communicate via the same message structure. In this implementation, a routine for the TI was provided that acts as an interface between the program that handles the I/O request and the DSR. Programs that still reside in the

TI continue to request console I/O by setting up a PRB and issuing an SVC, even though now the program providing the function resides and executes on a remote processor. The distributed communication routine sees to it that all messages are delivered or that the sender is notified of the hardwae failure which must have occurred. The communication routine in the TI, COMDSR, builds a message with the control information from the PRB (and the text on output) and sends it to the SMS function module. A communication routine in the module performs the send and receive functions on messages. Note that the function module uses only messages for communication with other modules. When a message arrives, the communication routine extracts the information, opcode, character count and text, and stores it locally for LOGDSR implemented in the SMS. When LOGDSR has information to pass to the calling program in the TI, it calls the communication routine which builds a message to send to COMDSR.

A program requesting I/O assumes a virtual path between its buffer and the device when, in fact, there is a multilevel protocol generated. Figure 3.8 illustrates these levels and an I/O request from a user program to the console device. The TI provides an SVC call to enable user routines to request services only available at the system level. This same mechanism has been used to implement the send and receive function on messages.

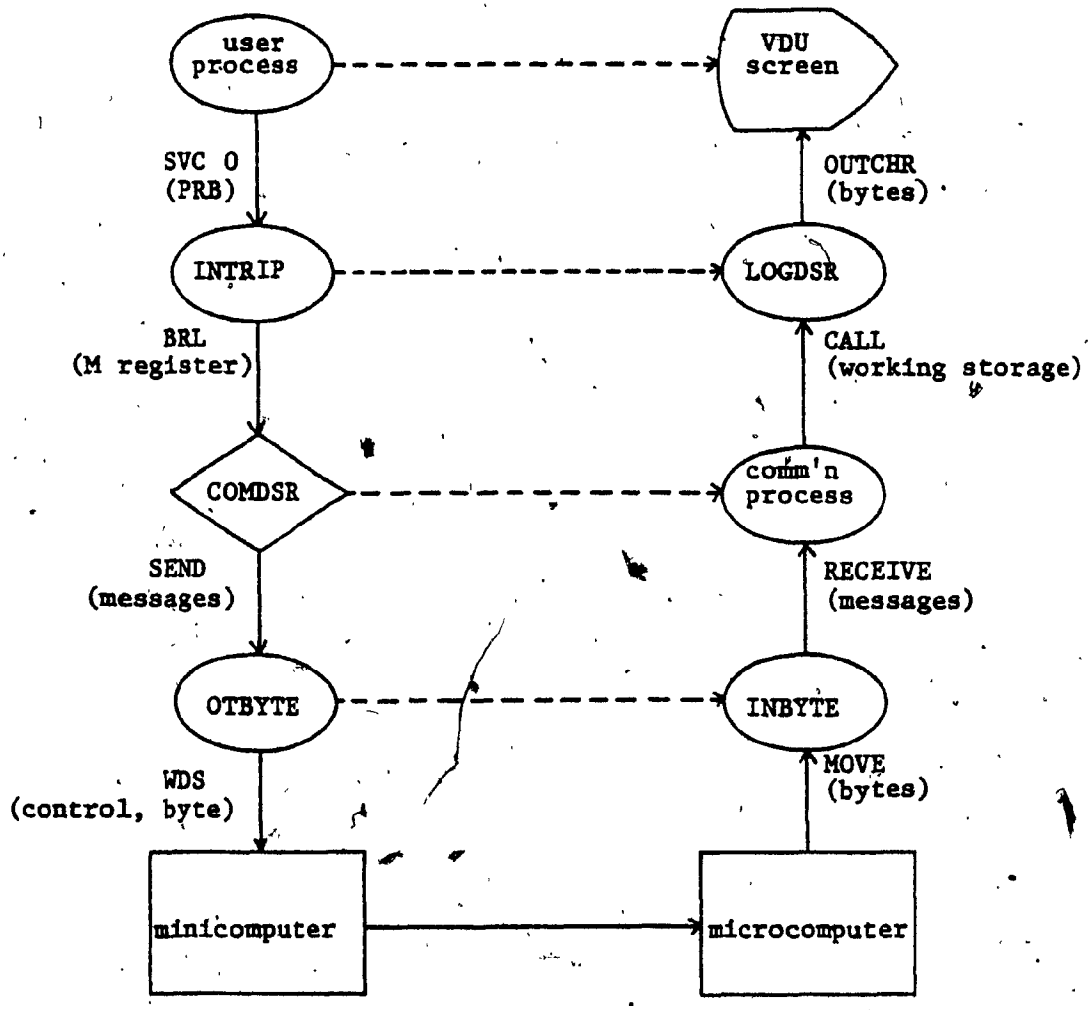


Figure 3.8 Multilevel Protocol for Message Transfer

3.3.2 MESSAGE FORMAT

All messages are made up of three principal parts, either implied or explicit: header, text, and end. The complexity of header information depends on and should be in reasonable balance with 1) the sophistication of the system application, and 2) the type of communication network over which the message is to be passed. A standard format able to meet the requirements of a complex system, as well as be compacted so that a subset of the basic header could be used for simpler systems has been proposed in White (1971). It is based on the efforts of the American National Standards Institute Task Group X3S33- Message Header Formats and has been used in this experiment as the guideline for the logical structure. Various message formats and protocols are outlined in Appendix C for comparison purposes and as a starting point for further investigation.

A header is divided into an address section, relating to the communication system, and a reference section, relating to the calling routine. A header section of sixteen items (the Link Date-Time Group is considered a part of the Link Message Identity) can handle even the most sophisticated network, a store and forward packet switching system. The convention is that the first word of the header, a Heading Item Indicator (HII), can suppress those items not required and, thereby, reduce unnecessary

overhead. In effect, it identifies the type and complexity of the message, for example, messages in MMCMS have an HII code of $0B91_{16}$ (see Figure 3.9 for how this was arrived at). This signifies a header made up of the following:

Destination address - information provided by the originator of the message identifying the station (or stations) to which the message is to be delivered,

Reference Station Identification - information supplied by the message originator to identify the program that performs communication servicing for it,

Originator Station Identification - identifies the address of the station from which the message was first entered into the system,

Originator Message Identification - distinguishes the message from others transmitted from the same station,

Programming Designator - information used to determine which program at the destination should be called to handle the message text, may include record length,

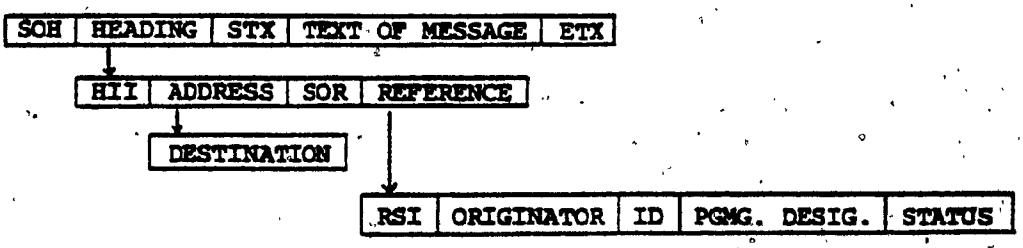
Message Status - added by the originator or communication subsystem to indicate the delivery status.

The text or body of the message contains the actual information (other than control) that is to be communicated. The principle for handling the text is that it must be delivered without change to the addressee. The end of the message is indicated by a unique end-of-text character.

HII/Heading Item Indicator	(0B91)
ADDRESS	
Link Message Identity/Date-Time Group	0
Link Message Status	0
Privacy/Classification	0
Precedence Indicator (per address)	0
Destination Address (per address)	1
Secondary Routing/Handling Information	0
REFERENCE	
Reference Station Identification	1
Originating Station Identification	1
Originating Message Identification	1
Originating Date-Time Group	0
Message Accounting Information	0
Programming Information:	
Program Designator	1
Program Precedence	0
Program Modifications/Options	0
Program Access Code	0
Message Status	1

0 signifies item is not currently being used
1 signifies item is present

Figure HII/Heading Item Indicator Coding



1) SOH/Start of Header (8 bits)
2) HII/Heading Item Indicator (16 bits)
3) Destination name (48 bits)
4) SOR/Start of Reference (8 bits)
5) RSI/Reference Station Identification (16 bits)
6) Originator name (48 bits)
7) Message Identification (8 bits)
8) Programming Designator: opcode (8 bits)
   error number (8 bits)
   record length (8 bits)
9) Message status (8 bits)
10) STX/Start of Text (8 bits)
11) Text of message (maximum of 2040 bits)
12) ETX/End of Text (8 bits)

Figure 3.10 Message Format

Using these guidelines not only gives us a message format that meets ANSI recommendations, but also the flexibility to modify or expand the format in future if experimental results warrant it. The message format is illustrated in Figure 3.10.

3.4 IMPLEMENTATION DETAILS IN TI HOST

Before the implementation could begin, a decision had to be made on how a message-based communication facility could be added to the current TI operating system. Examining the flow of control between a user program and the operating system shows that the boundary between the two is crossed via an SVC call and that INTRIP is the first, or highest level routine in the operating system that is executed (Figure 3.11). Since INTRIP already handles all requests for I/O, it was logical to insert the send and receive function for messages at this level.

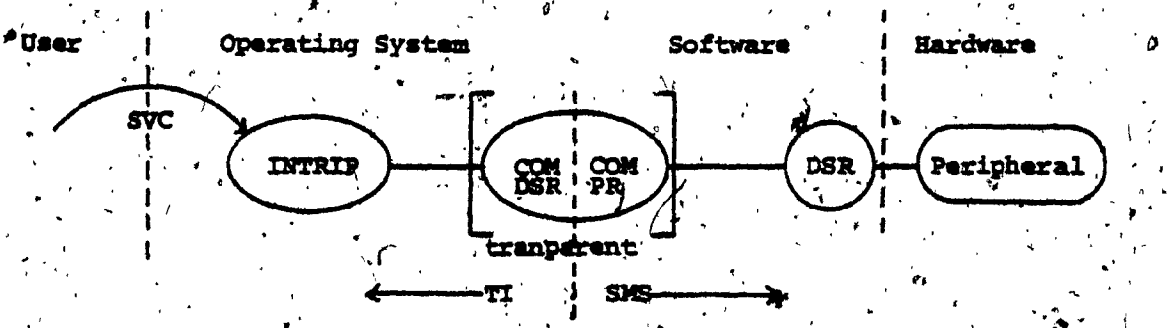


Figure 3.11 Implementation of Communication Routine

This was implemented as a separate communication routine interposed between INTRIP and the device service routines.

Its responsibility would be to accept control information formatted in the form of a PRB and transform it to the message format, perform the explicit data exchange, extract the PRB information and text at the destination, and call the DSR. In this way, it is completely transparent to its neighbours, INTRIP and the DSR.

At the TI end, COMDSR handles the communication link and interfaces to the operating system exactly like other DSRs. It is linked to the operating system through the use of the system tables. The program is associated with a new physical device, COM, representing the physical interconnection path, through a PDT entry. Thus it can be invoked by the operator or operating system by assigning a logical unit to COM. This has the effect of initializing a message based communication path to the remote processor and transferring control of the peripheral device to the microprocessor based function module. The advantage of this solution is that all modifications to the host system are contained within a single module.

On an I/O request, control is passed from INTRIP to COMDSR which builds a correctly formatted message from the information in the PRB and does the send (addressed by name) to the device. On output, the record length, passed in the PRB, is checked to ensure it is positive and within limits (currently a maximum of 800 bits of text can be sent in one message). All other checking pertains to the I/O device and

is done in the DSR. When a Request For Next Message (RFNM) is received from the remote processor, the status information is extracted and plugged in the PRB and system tables. For example, bit 3 of LDT(0) has to be set on an open and reset on a close and PDT(4) has to be plugged with the error number, since these tables are still local to the TI and inaccessible to the SMS. If the destination name does not match any valid device name, then an error message is printed through MSGSM, the system message writer.

Included for testing is the ability for COMDSR to simulate the response from the remote function module, and thereby, interact with itself. This was used to test the hardware and communication protocol from the TI end before the SMS was operational. In the current implementation, the same physical link and interface is used for interprocessor communication as for loading SMS program storage (Figure 3.13). Consequently, routines in SMSUTL (written for the development package and described in section 3.5) are used for performing the word by word I/O. With a different physical link, new interface routines would be written to enable COMDSR to handle the link. This implies that the message communication layer could be mapped onto various physical interconnections.

The code and structure of COMDSR, because it provides the software interface to the link via a standard 16 I/O Data module, is very similar to other DSRs. COMDSR,

including the appropriate parts of SMSUTL, requires about 450 words of assembler code, while a typical DSR comes to about 400 words. The message header format and error messages account for the slightly higher memory requirements. This suggests that quite a substantial saving in TI memory could result as other DSRs migrate out to computer modules. The assembler listing of the COMDSR routine has been included in Appendix B, whereas SMSUTL has been included with the development programs in Appendix D.

In the present implementation, only one message can be in transit at any one time. Provision has been made, however, to allow modules to send a number of messages before receiving a reply. Each message has been assigned a unique number (0 to 255) along with its originator's name so as to be able to differentiate between messages (this could also be done with a date-time stamp). This will enable multiple conversations when other functions are distributed in the next phase of implementation.

3.5 IMPLEMENTATION DETAILS OF SMS FUNCTION-MODULE

Since there is no provision in the SMS for linking separate routines, the software consists of one main program, TLOG, made up of a number of routines:

1) a simple control program whose job is to initialize local variables and monitor the

communication medium, waiting for a message to arrive. When a message arrives, the destination field is checked against its own name, in this case, LOG. On a match, control is passed to the communication routine. Other messages are ignored.

2) a communication routine which provides the send and receive functions for the SMS implementation of the function module. It extracts the control information from the message header and the message text, if it is present, for use by the DSR. On return from LOGDSR, the status information is built into a return control message and sent to the originator.

3) a device service routine whose responsibility is to provide the same functions that the user is aware of in the TI implementation of the DSR.

4) utility procedures to perform I/O of one byte between SMS working storage and the communication link or the console log.

The code for the communication routine, DSR, and subroutines takes approximately 800 words of program storage. The assembler listing for the routines that create a function module of the SMS can be found in Appendix E.

3.6 DISTRIBUTED CONTROL FLOW

Communication between a program in the TI and the DSR implemented in the SMS requires an explicit data exchange in an agreed upon format. Programs in the two computers can run in parallel, where their synchronization is dependent on the receiving of messages. Once again, the system utility program, COPYAL, will be executed as the example. Card images will be copied to the console log, so that the new communication path and protocol will be used. This demonstrates the third level of implementation, where a function is provided by an interconnected hardware module. The program flow with distributed software, the DSR external to the TI, is diagrammed in Figure 3.12. In this case, the output device must be assigned to the communication medium:

```
A,28,CR)
A,27,COM)
```

The ASSIGN command links the LDT for logical unit 27 to the PDT of the communication medium.

```
COPYAL)
```

Whenever COPYAL issues an I/O request, via a PRB and SVC call, the instruction is trapped by INTRIP just as before. The same processing is done, including a check that LUN 27 has been assigned. INTRIP then links to the PDT via LDT(2) and, finds the address of the routine that handles the request, in this case, COMDSR. The communication routine has responsibility to build

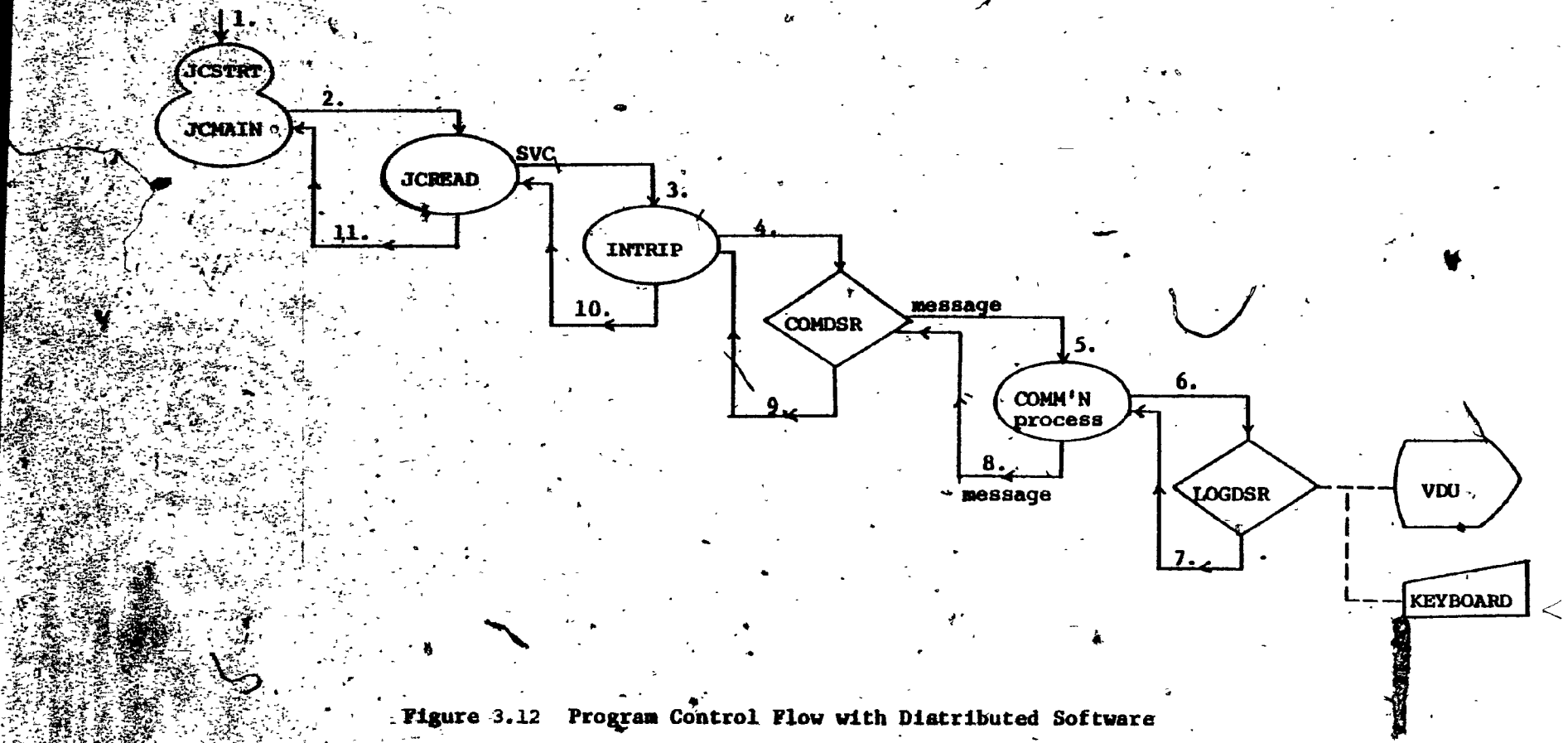


Figure 3.12 Program Control Flow with Distributed Software

information from the system tables, PRB, and local variables. It then sends the message, made up of the header just constructed, text on output, and end character on the communication medium to the SMS. It then waits for a return message, all the while monitoring the communication medium. Whenever the SMS is not actively doing the I/O, it monitors the communication medium checking for the arrival of a message. When it finds one, control is passed to the communication routine which removes the pertinent information, that is, some fields from the header and the complete text, and stores it in working storage. Control is then passed to LOGDSR, the device service routine for the console log implemented in the SMS. On completion of the request specified by the opcode, status is returned to the communication routine. A return message, containing status and, on input, the number of characters read and text, is sent to the originator of the message in the TI. The SMS control program then goes back to monitoring the communication medium. When COMDSR recognizes that a message has arrived, it checks that it is the message it is waiting for. The communication routine gets the status and updates the caller's PRB, LDT, if required, and, on input, transfers the message text to the caller's buffer. Control is returned to INTRIP which returns to the caller, in our example, COPYAL. This procedure is repeated until an end-of-file is encountered on the card reader.

3.7 PROGRAM DEVELOPMENT AND LOADING

The University computer centre's CDC Cyber/172, the TI 980B, and custom designed hardware were all used as development tools.

SMS supplies a cross assembler (MCCAP) written in Fortan, which includes a macro capability, automatic procedure handling, and conditional assembly. MCCAP was installed on the CDC machine since it requires random access I/O for Fortran which does not exist in the current TI minicomputer software. Using the CDC proved advantageous as its availability, multi- user environment, and software support helped in the creation and modification of SMS programs.

After a program had assembled correctly, it was transferred from the CDC to the TI development system. This was done by loading an interrupt driven, special purpose operating system (COMSYS) that makes the TI appear as a TTY terminal to the CDC host. The TI is connected like a regular terminal, via an acoustic coupler and 1200 baud line. A routine at the CDC end (SEND) performs the transfer of a local file to the TI disc (this entire procedure is outlined in Gillespie, 1977b).

The SMS object file is then downloaded from the TI disc directly to SMS program storage by a TI utility called

LODMEM. Two features of LODMEM are worth mentioning:

1) To ensure the reliability of the object code in program storage, each byte output is checked by doing two reads and comparing them against what was output. If there is a mismatch, the operation is repeated. After ten tries an error message is output to the console.

2) Since LODMEM inputs ASCII characters (an octal address followed by up to eight instruction codes), it can also be used to modify SMS code in memory from the keyboard.

LODMEM has to issue RDS and WDS privileged instructions to set the address on the interface board and read and write the 16-bit binary instruction codes. Therefore, a utility package called SMSUTL was written to provide these function and generated as part of the operating system software. It contains the following routines to support LODMEM and, for the time being, the communication protocol:

OTADR- checks that the address is in range, outputs a 16-bit address as upper and lower bytes

OTINST- outputs a 16-bit instruction to the preset address in two bytes, masks the idiosyncracies of the hardware

CHEQ- tries to ensure that the byte of the instruction just written is correct by doing two reads and

comparing them to what was written

OTBYTE- checks that the address is valid, writes one byte, does wraparound on the TI-SMS shared memory buffer

INBYTE- checks that the address is valid, reads one byte, does wraparound on the TI-SMS shared memory buffer

OUTP- outputs a block of data from TI, does unpacking

INP- inputs a block of data to TI, does packing

LODMEM communicates with the operating system to OTADR and then OTINST through the standard technique of an SVC call. INTRIP was modified to recognize SVC 7 and 8 as calls to these routines.

3.7.1 HARDWARE DETAILS

Since the SMS300 comes from the manufacturer with a 2K ROM chip, a 1K fast RAM was interfaced to the system for program development. An interface card was designed with two connectors, one to attach to the SMS bus and one for a cable to a 16 I/O data module in the TI. The physical setup is represented in PMS notation in Figure 3.14. These two boards were developed in conjunction with a general microprocessor development system, which provides a complete display and control panel, and in fact, were tested on this system. However, because the system was designed and built for 8-bit microprocessors before the SMS was selected, it

does not support SMS development. A separate system was built for the SMS and the interface and memory cards borrowed from the development system. Since the SMS has an instruction length of 16 bits, it requires two writes to the interface to load one instruction. At the software level, the program LODMEM, in conjunction with the routines in SMSUTL, allows loading an instruction by specifying one address and the instruction code.

3.7.2 DISPLAY AND CONTROL

A simple front panel of switches and LEDs was provided to display and control the SMS. There are three switches: a RUN/HALT switch, a TI access/SMS access of program storage with memory protect, and a reset switch. There is a LED hexadecimal display of the memory address register and the instruction. A small display board, consisting of 8 toggle switches and 8 LEDs, can be connected to one of the IV bytes, so that with software support in the SMS, registers and working storage locations can be set or displayed. It is conceded that these facilities are very primitive, but they were found to be adequate.

To add display capability, a dump program was implemented in the TI as a job control command:

DM[PMEM],memory type,first address,last address

The parameter 'memory type' allows the dumping of TI memory,

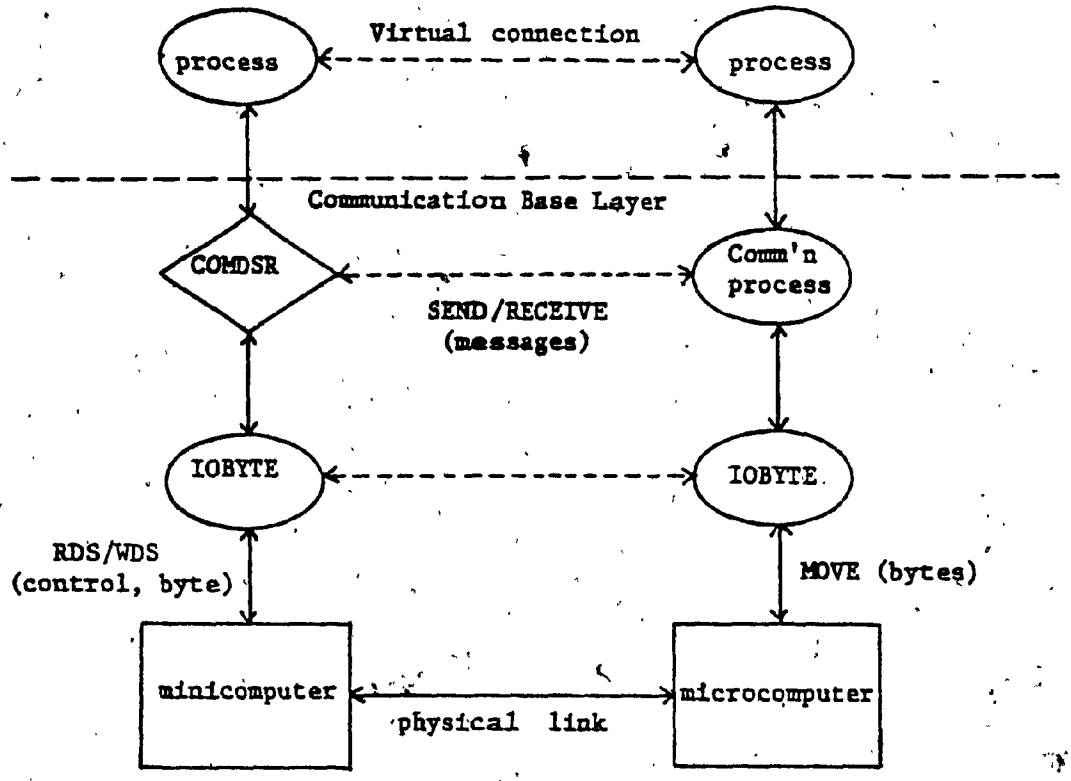


Figure 4.2 Communication Base Layer

In the current implementation, only one function of a uniprogramming operating system, the console log device service routine, was distributed to a computer module. All other routines used the TI minicomputer as their host. The implementation demonstrates that the interaction between INTRIP (as the agent for a program) and the LOGDSR function module, programmed in the SMS300, works correctly. Furthermore, the SMS module can provide the same functions as the DSR implemented in the TI.

As a next step, the second level of control, the job control monitor which inputs the command, scans it for parameters, and decodes it, could also be distributed. JCMAIN currently requires almost 400 words to perform its functions, but 125 words are used for buffers (which would use working storage as opposed to program storage) and 51 words perform the linkage to the job control programs (which would not be implemented in the SMS module but in COMDSR). With an optimization to the code already written (800 words), the functions of JCMAIN could be added to the SMS module to fit within the 1K memory limit. The addition of JCMAIN to the LOGDSR module would reduce transmission overhead. It can guarantee that the command inserted in the message is syntactically correct and, in this way, reduce traffic on the link joining the device to the main processor. Alternatively, it can decode the job control record, build a message containing the parameters, and send it directly to the module implementing the command. This is shown in the diagram of Figure 4.3 where COMDSR accepts the return message, delivers it to the requested job control module, and then passes control to it.

The logical continuation of this project is the implementation of other DSRs in the operating system to a set of interconnected computer modules. Different microprocessors should be examined as possible candidates for the implementation of computer function modules. With

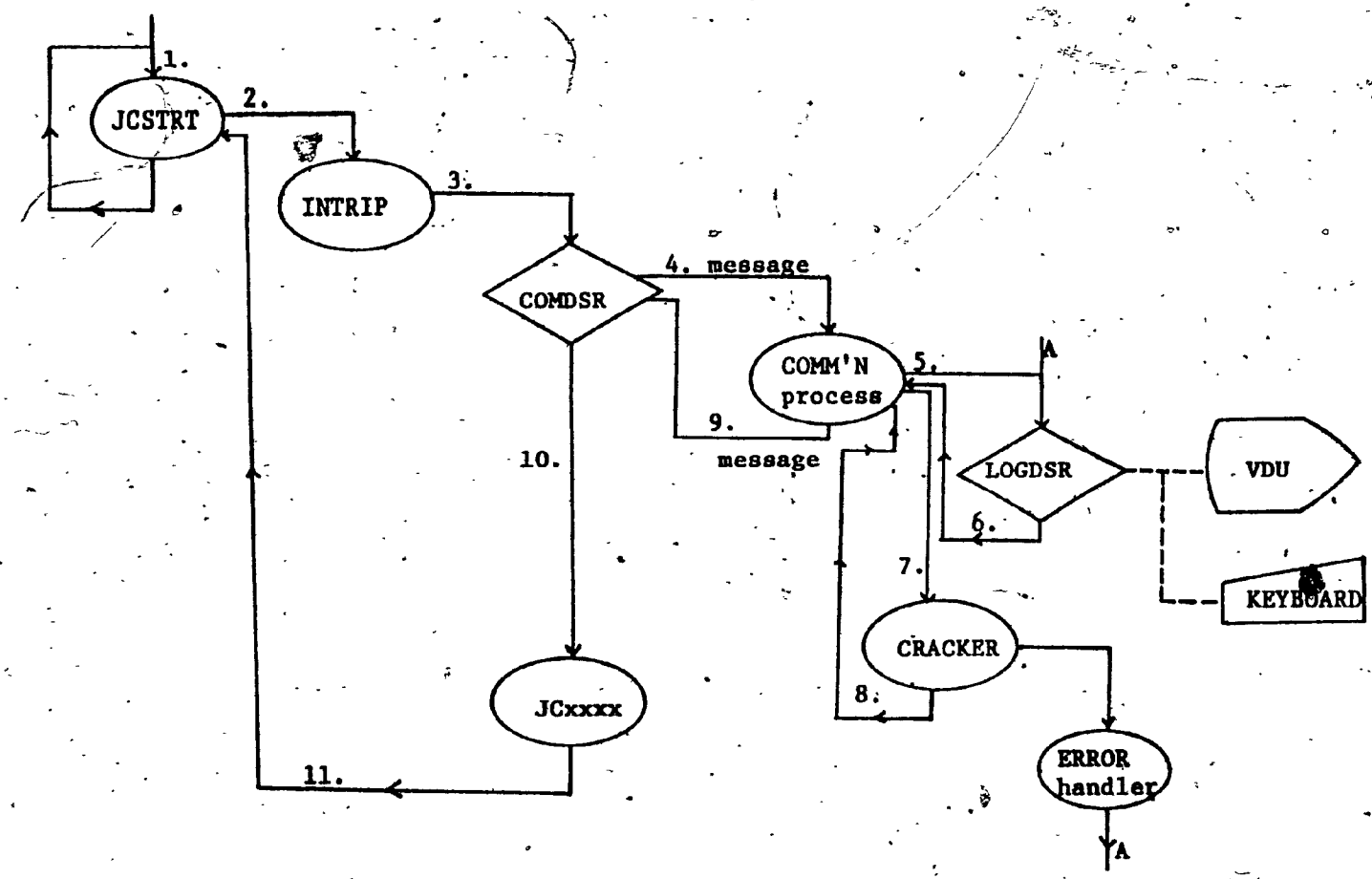


Figure 4.3 Program Control Flow with JCMAIN Functions Distributed

the experience of having implemented different DSR functions and a communication routine in different microprocessors, recommendations could be made for necessary and optimum characteristics of CM modules.

Once the first level of distribution has been performed, other OS functions could begin to migrate to function modules. The second level would add another dimension to the problem since the interconnections and control flow would be more complex. At this level, the direction will shift from decomposing an operating system to creating and designing one from a set of interconnected function modules. The operating system, architecture, and distribution would be defined concurrently, at the same level.

The results of this work have demonstrated a functional decomposition of a uniprogramming operating system at the DSR level. Multiprogramming systems organized as a set of cooperating processes should be examined since they stand to gain the most by distribution to function modules. This can be accomplished through the provision of a communications facility as outlined above which is capable of handling the transfer of control and data information between the component computers.

ANNOTATED BIBLIOGRAPHY

Abrams, M.D. and P.G. Stein, Computer Hardware and Software: An Interdisciplinary Introduction, Addison-Wesley, Reading, Massachusetts, 1973.
[examines both hardware and software in computer systems, tradeoffs, references and questions at end of each chapter]

Abramson, N. and F.F. Kuo, Computer-Communication Networks, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
[description of Dartmouth Timesharing, IBM TSS, ARPA, and ALOHA systems, also general issues in networks]

Akkoyunlu, E., A. Bernstein, and R. Shantz, "An Operating System for a Network Environment", Proceeding Symposium on Computer-Communication and Teletraffic, April 1972, pp. 529-538.
[cooperating processes on layered, abstract machine, hierarchical, timesharing OS, location transparency and homogeneous IPC using ports]

Alsberg, P.A., "Intelligent Terminals as User Agents", IEEE 1976 Trends and Applications: Micro and Mini Systems, May 1976, pp. 129-135.
[microprocessor builds commands from touch input, application oriented prompts, displays only valid options]

Allison, D.R., "Software Issues in LSI Microprocessor Design", Compcon 76 Digest of Papers, 12th IEEE Computer Society International Conference, February 1976, pp. 15-18.
[hardware constraints of LSI, high level languages, analysis of programs, process models, data structures]

Anderson, G.A. and E.D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", ACM Computing Surveys, Volume 7, Number 4, December 1975, pp. 197-213.
[provides a starting point and common ground for discussion of interconnection schemes, describes configurations of Processing Elements, paths, and switching elements]

Arden, B.W. and A.D. Berenbaum, "A Multi-microprocessor Computer System Architecture", Proceedings 5th Symposium on Operating Systems Principles, November 1975, pp. 114-121.
[OS functions in hardware processors, very large, slow memories with small cache, processor clusters connected by loop]

Atkinson, J., "Architecture of Series 60/Level 64", Honeywell Computer Journal, Volume 8, Number 2, 1974, pp. 94-106.

Balzer, R.M., "PORTS - A Method for Dynamic Interprogram Communication and Job Control", Proceedings Spring Joint Computer Conference, Volume 38, May 1971, pp. 485-489.

[unified communication between a program and files, peripherals, other programs, supervisor, port can be forcing a hierarchy]

Baskett, F. et al., "Task Communication in DEMOS", Proceedings 6th Symposium on Operating System Principles, SIGOPS, 1977, pp. 23-32.
[description of links for communication in uniprocessor CRAY-1]

Bell, C.G. and A. Newell, "The PMS and ISP Descriptive Systems for Computer Structures", Proceedings Spring Joint Computer Conference, Volume 36, May 1970, pp. 351-374.
[first and definitive paper on PMS and ISP notations, reasons for their development, uses, ideas]

Bell, C.G. and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, New York, 1971.
[definition, reasons, uses, philosphy behind PMS and ISP notation, current computer systems described and examined using the notations]

Bell, C.G., M. Knudsen, and D. Siewiorek, "PMS: A Notation to Describe Computer Structures", Compcon 72 Fall Digest, 6th Annual IEEE Computer Society International Conference, September 1972, pp. 227-230.
[concise description of PMS notation, examples of how it has been used]

Bell, C.G. et al., "The Architecture and Applications of Computer Modules: A Set of Components for Digital Systems Design", Compcon 73 Digest of Papers, 7th Annual IEEE Computer Society International Conference, February 1973, pp. 177-180.
[a sophisticated module for building distributed systems is introduced, discussion of need for efficient synchronization and communication]

Bell, G. and W.D. Strecker, "Computer Structures: What We Have Learned from the PDP-11", Proceedings 3rd Annual Symposium on Computer Architecture, 1975, pp. 1-14.
[review of successes and shortcomings of PDP-11 design, can be extrapolated to other minicomputers]

Brinch Hansen, Per, "The Nucleus of a Multiprogramming System", Communications ACM, Volume 13, Number 4, April 1970, pp. 238-241.
[dynamic creation and control of hierarchy of parallel, cooperating processes]

Brinch Hansen, Per, RC4000 Software: Multiprogramming System, 2nd edition, Copenhagen, A/S REGNECENTRALEN, 1971.
[manual of structured operating system, readable, detailed, processes communicate via messages in pipes]

Brinch Hansen, Per, Operating System Principles, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
[theme is that operating systems are not unlike other programs, discusses resource management and synchronization]

Brinch Hansen, Per, "The SOLO Operating System: A Concurrent Pascal Program", Software-Practice and Experience, Volume 6, 1976, pp. 141-149.
[description of a single user operating system from the user's point of view]

Brown, G.E., R.H. Eckhouse, Jr. and R.P. Goldberg, "Operating System Enhancement through Microprogramming", SIGMICRO, Volume 7, Number 1, March 1976, pp. 28-33.
[real-time, process structured operating systems, criteria for determining functions for implementation in firmware, advantages]

Browns, S. and T. Fancott, "Microprocessor Implementation of a Distributed Software Function", Proceedings of the International Symposium on Mini and Micro Computers, November 1977, pp. 136-140.
[experimental implementation of a device service routine for operator console, investigation of interprocessor communication, partitioning of OS along functional lines]

Chesson, G. L. "The Network Unix System", Operating Systems Review (Special Issue), November 1975, pp. 60-66.
(also Proceedings 5th Symposium on Operating System Principles, November 1975)

Coffman, Jr., E.G. and P.J. Denning, Operating Systems Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
[formal study of synchronization, deadlock, scheduling, memory management]

Coker, C.H., "An Experimental Interconnection of Computers through a Loop Transmission System", Bell System Technical Journal, Volume 51, Number 6, July-August 1972, pp. 1167-1175.
[results and analysis of two DDP 516 minicomputers connected to Pierce loop]

Cooper, R.G., "Micromodules: Microprogrammable Building Blocks for Hardware Development", Proceedings 1st Annual Symposium on Computer Architecture, December 1973, pp. 221-226.
[small family of modules, standardized interconnection discipline, design aids]

Coury, F.F., "A Systems Approach to Minicomputer I/O", Proceedings Spring Joint Computer Conference, Volume 36, May 1970, pp. 677-681.
[almost humorous if you can laugh at yourself, makes the point that problem solver-computer interface is crucial]

Cox, G.W. and W.B. Schneider, "On Improving Operating System Efficiency through Use of a Microprogrammed, Low-level Environment", MICRO7 - Seventh Annual Workshop on Microprogramming, September 1974, pp. 297-298.
[only measurements of performance should decide between firmware and software implementations]

Cypser, R.J. Communications Architecture for Distributed Systems, Addison-Wesley Publishing Co., Reading, Massachusetts, 1978.
[textbook on IBM's System Network Architecture (SNA)]

Davies, D.W. and D.L.A. Barber, Communication Networks for Computers, John Wiley and Sons, London, 1973.
[problems concerning public data networks, data transmission methods, error control, SITA, ARPA, NPL networks]

Dijkstra, E.W., "The Structure of the 'THE' Multiprogramming System", Communications ACM, Volume 11, Number 5, May 1968, pp. 341-346.
[classical paper on synchronization in an operating system, semaphores, author's experiences in building a layered system]

Dromard, F., "Design of a Microprogrammed Alphanumeric Terminal", MICRO7 - 7th Annual Workshop on Microprogramming, September 1974, pp. 128-134.
[syntactical filtering to reduce dialogue overhead, used with conversation programming of simplified FORTRAN]

Ellis, R.A. "Modular Computer Systems", Computer, October 1973, p. 13.
[introduction to issue on modular hardware]

Emmons, W.F., "Data Network Protocol Standards", Proceedings of Computer Networking Symposium, December 1977, pp. 9-17.
[history, ANSI standards for message formats, three general types of headings proposed, independent of communication path]

Falk, H., "Computer Report I - Hard-Soft Tradeoffs", IEEE Spectrum, February 1974, pp. 34-39.
[shift towards hardware implementation, list of possible candidates, brief description of 2 hardware processors]

Fancott, T. and W.G. Probst, "Software Distribution in a Microcomputer-Based Multiprocessor", Proceedings 6th Texas Conference on Computer Systems, November 1977, pp. 4B-28-4B-34.
[extension of modular, message based operating system functions distributed to interconnected microcomputers, philosophy and preliminary design]

Farber, D.J. and K.C. Larson, "The Structure of a Distributed Computing System - Software", Proceedings Symposium on Computer Communications Networks and Teletraffic, April 1972a, pp. 539-545.
[description of operating system, heterogeneous ring network, fail-soft, message based communication]

Farber, D.J. and K.C. Larson, "The Structure of a Distributed Computing System - The Communication System", Proceedings Symposium on Computer Communications Networks and Teletraffic, April 1972b, pp. 21-27.
[loop, ring interfaces, associative addressing by process name]

Farmer, W.D. and E.E. Newhall, "An Experimental Distributed Switching System to Handle Bursty Computer Traffic", Proceedings ACM Symposium on Data Communications, October 1969, pp. 1-33.

[loop with variable length messages, hardware description, data transparency of messages]

Forsdick, H.C., R.E. Schantz, and R.H. Thomas, "Operating Systems for Computer Networks", Computer, Volume 11, Number 1, January 1978, pp. 48-57.
[discusses NOS as method of realizing full potential of networks, specific references to RSEXEC system to couple PDP-10 hosts under TENEX and ARPANET, and NSW to provide access via simple uniform means to a wide variety of development tools on different hosts]

Foster, C.C., "A View of Computer Architecture", Communications ACM, July 1972, Volume 15, Number 7, pp. 557-565.
[look 25 years into future, effects of microcomputers in non-computational areas, super computers reconsidered]

Fraser, A.G., "A Virtual Channel Network", Datamation, February 1975, pp. 51-53, 56.
[Spider system, eleven minicomputers connected to loop by TIUs, packet switching]

Fuller, S.H. and D.P. Siewiorek, "Some Observations on Semiconductor Technology and the Architecture of Large Digital Modules", Computer, October 1973a, pp. 15-21.
[workshop report, RT level modules, PMS modules, communication between hardware modules, description of C.mmp, PLURIBUS, CMs interconnection]

Fuller, S.H., D.P. Siewiorek and R.J. Swan, "Computer Modules: An Architecture for Large Digital Modules", Proceedings 1st Annual Symposium on Computer Architecture, December 1973b, pp. 231-237.
[detailed look at CMs and multilevel interconnection single word transfer, hardware address mapping]

Gillespie, W. and G. Mack, "Microprocessor Support System-Cross Assemblers", Department of Computer Science, Concordia University, 1977.
[includes documentation of SMS cross assembler on CDC]

Gillespie, W., "Microprocessor Support System", Department of Computer Science, Concordia University, 1977.
[description of how to use COMSYS, details of other software not appropriate to SMS]

Gray, M.T., "Microprocessors in CRT Terminal Applications: Hardware/Software Tradeoffs", Computer, October 1975,
[for polled terminals using complex protocols, text editing, error handling]
[ARPA subnet, message packet formats, description of IMP hardware and software, reliability, recovery]

Greenblott, B.J. and M.Y. Hsiao, "Where is Technology Taking us in Data Processing Systems?", Proceeding National Computer Conference, Volume 44, May 1975, pp. 623-628.
[both LSI and magnetic recording technology are discussed, effect on user-terminal based systems]

Habermann, A.N., "Synchronization of Communicating Processes", Communications ACM, Volume 15, Number 3, March 1972, pp. 171-176.
[formal description of a system that sends and receives messages in buffers, proof of synchronization]

Hassing, T.E. et al., "A Loop Network for General Purpose Data Communications in a Heterogeneous World", Proceedings Symposium Data Communications, November 1973, pp. 88-96.

[packet switching in interconnected loops, formats, ring interface, network protocol, and security discussed]

Hastings, A. and D. Kozlay, "A Software Development Instrument", Compcon 75 Fall Digest, 11th IEEE Computer Society Conference, September 1975, pp. 201-204.
[interactive debug aid and FORTRAN compiler which runs on Intel 8080 microprocessor]

Heart, F.E. et al., "The Interface Message Processor for the ARPA Computer Network", Proceedings Sprint Joint Computer Conference, Volume 36, May 1970, pp. 551-567.
[IMP subnet, protocols, acknowledge via RFNM, interface with Hosts, hardware and software described, reliability, recovery procedures]

Heart, F.E., "A New Minicomputer/Multiprocessor for the ARPA Network", Proceeding National Computer Conference, Volume 42, June 1973, pp. 529-537.
[configure as many Pcs as required to meet throughput, uses shared and private memory]

Hoare, C.A.R., "Operating Systems: Their Purpose, Objectives, Functions and Scope", Operating System Technique, Hoare and Perrott (eds.), Academic Press, London 1972, pp. 11-19.
[philosophical yet practical, views and historical development of operating systems]

Hoare, C.A.R., "Monitors: An Operating System Structuring Concept", Communications ACM, Volume 17, Number 10, October 1974, pp. 549-557.
[in Hoare's inimitable style, critical regions and shared data are combined in monitor concept]

Hodges, D.A., "Trends in Hardware Technology", Computer Computer Design, February 1976, pp. 77-85.
[examination of progress in processor, memory, I/O, communication, technology, implications for computer systems]

Hopper, G.M., "Technology: Future Directions", Proceedings of Conference '74 Data Processing Institute, June 1974, pp. 251-256.
[discusses importance of standards, high-level programming and hardware for meeting future needs]

Jafari, H., T. Lewis and J. Spragins, "A New Ring-Structured Microcomputer Network", Proceedings International Conference on Computer Communications, September 1978, pp. 167-174.

Jasper, D.P., "Position Paper on Usability for Data Services Why Not a Mini-computer in Every Cupboard)", Compcon 75 Fall Digest, 11th IEEE Computer Society Conference, September 1975, pp. 65-69.
[points out advantages of service networks, minis and maxis, ideas on system usability]

Jones, A.K. et al., "StarOs, a Multiprocessor Operating System for the Support of Task Forces", Proceedings 7th Symposium on Operating System Principles, December 1979, pp. 117-127.

[operating system designed for use with CMs, uses message based system on mailboxes for communication and synchronization, first operating system for Cm*, emphasizes facilities rather than structure (see Ousterhout)]

Joseph, Earl C., "Future Computer Architecture-Polysystems", Compcon 72 Fall Digest, 6th Annual IEEE Computer Society International Conference, September 1972, pp. 149-153.
[well-reasoned view of one possibility of future systems, macro-modules to implement software functions and create architectures]

Joseph, E.C., "Innovations in Heterogeneous and Homogeneous Distributed-Function Architectures", Computer, March 1974, pp. 17-24.
[survey article, quite complete but not in depth, unfortunately no bibliography]

Joseph, E.C., "Computers and Networks 1980 - Some Architectural Trends", Compcon 76 Digest, 12th Annual IEEE Computer Society International Conference, February 1976, pp. 69-72.
[architectural alternatives, hard programs, distributed intelligence, compilable architectures]

Kahn, R.E., "Resource-Sharing Computer Communications Networks", Proceedings IEEE, November 1972, pp. 1397-1407.
[ARPANET, sharing of data bases, programs, hardware via a packet-switching network]

Kildall, G.A., "High-Level Language Simplifies Microcomputer Programming", Electronics, June 27, 1974, pp. 103-109.
[PL/M for Intel MCS-8 family of micros, cross compiler]

Korn, G.A.,Minicomputers for Engineers and Scientists, McGraw-Hill, New York, 1973.
[describes minicomputers from application viewpoint, discusses architectures, interfacing, and peripherals]

Knuth, D.E., "An Empirical Study of Fortran Programs", Software - Practice and Experience, Volume 1, Number 2, April/June 1971, pp. 105-171.
[study of Fortran programs to discover what instructions programmers use, implications to compiler design]

Kropfl, W.J., "An Experimental Data Block Switching System", Bell System Technical Journal, Volume 5, Number 6, July-August 1972, pp. 1147-1165.
[hardware implementation of ring interface and loop supervisor, message formats, hog prevention technique, bypass box]

Lee, Imsong, "LSI Microprocessors and Microprograms for User-Oriented Machines", MICRO 7 - 7th Annual Workshop on Microprogramming, September 1974, pp. S1-S13.
[stimulating description of hardware/software tradeoffs, LSI technology, and personal computing]

Lewis, T.G., "How Large Should a Computer Be", SIGMINI Newsletter, Volume 2, Number 1, January 1976, pp. 12-16.
[mathematically shows tradeoff point for multiprogramming and multiprocessing, stresses modularity]

Liskov, B.H., "The Design of the Venus Operating System", Communications ACM, Volume 15, Number 3, March 1972, pp. 144-149.
[experimental multiprogramming system on small microprogrammable computer, modifications to architecture affecting operating system]

Liu, M.T and C.C. Reames, "Message Communication Protocol and Operating System Design for the Distributed Loop Computer Network (DLCN)", Proceedings 2nd Symposium on Computer Architecture, March 1977, pp. 193-199.
[integration of hardware, software and loop communication, bit oriented message protocol, automatic hardware acknowledge, error handling]

Madnick, S.E. and J.J. Donovan, Operating Systems, McGraw-Hill, New York, 1974.
[standard text, IBM oriented, annotated bibliography, seem to be errors in text]

Mandell, Richard L., "Hardware/Software Trade-offs - Reasons and Directions", Proceedings Fall Joint Computer Conference, Volume 41, December 1972, pp. 453-459.
[presents tradeoffs as fundamental to computer design, good examples, trades between I/O and processor]

Manning, E.G. and R.W. Peebles, "A Homogeneous Network for Data Sharing - Communications", Computer Networks, Volume 1, pp. 211-224.
[message switched subnet, loop or packet switched, all data (including messages) are segments, concept of locality of reference]

Marazas, G., "Microprocessors Heed the Call to Supervise I/O", Electronics, February 1978, pp. 104-108.
[minicomputer I/O, required characteristics of micros, distributed I/O boundary]

Martin, J., Teleprocessing Network Organization, Prentice-Hall, Englewood Cliffs, New Jersey, 1970.
[chapter 5 on error detecting codes, with examples, understandable]

Metcalfe, R.M., "Strategies for Interprocess Communication in a Distributed Computing System", Proceedings of the Symposium on Computer-Communication and Teletraffic, April 1972, pp. 519-526.
[uniform treatment of processes, concept of thin-wire connection, should also be used in OS design]

Mohan, C., "Survey of Recent Operating Systems Research, Designs and Implementations", ACM Operating Systems Review, Volume 12, Number 1, January 1978, pp. 53-89.
[broad survey, discusses impact of distributed processing on operating systems, interprocess communication proposals, modifications to uni-processor operating systems to work on multiprocessor systems and in a network environment]

Needham, R.M. and D.F. Hartley, "Theory and Practice in System Design", Proceedings ACM 2nd Symposium on Operating System Principles, October 1969, pp. 8-12.

[stresses importance of implementation and system tuning of operating systems as well as theoretical design]

Nelson, J.C., "The Economic Implications of Microprocessors on Future Computer Technology and Systems", Proceedings National Computer Conference, Volume 44, May 1975, pp. 629-632.
[developments in semiconductor technology, intelligent peripherals, importance of standards, partitioning by function]

Noyce, R.N., "Microelectronics", Scientific American, September 1977, pp. 62-69.
[history of development, technological breakthroughs]

Ousterhout, J.K. et al., "Medusa: An Experiment in Distributed Operating System Structure", Communications ACM, Volume 23, Number 2, February 1980, pp. 92-105.
[operating system structure closely matches hardware, Cm* multimicroprocessor, system made up of utilities, single task force, communicate via messages in pipes]

Parnas, D.L., "On the Criteria to be used on Decomposing Systems into Modules", Communications ACM, Volume 15, Number 12, December 1972, pp. 1053-1058.

Perez, A., D. Banerji, and J. Raymond, "Microprogrammed Operating Systems: A Design and Implementation Proposal", Computer Science Department, University of Ottawa, 1975, pp. 1-17.
[rough draft, ideas on what parts of operating systems could be implemented in microcode]

Pierce, J.R., "Network for Block Switching of Data", Bell System Technical Journal, Volume 51, Number 6, July-August 1972, pp. 1133-1145.
[network of closed loops for transmission of addressed blocks of data]

Poujoulat, G.H., "Architecture of the CORAIL Building Block System", Proceedings 4th Annual Symposium on Computer Architecture", March 1977, pp. 201-204.
[modular elements to build system that matches application, functional decomposition, micro implementation]

Pouzin, L., "Presentation and Major Design Aspects of the Cyclades Computer Network", DATACOMM 73, ACM/IEEE 3rd Data Communications Symposium, November 1973, pp. 80-87.

Probst, W.G. and G.V. Bochmann, "Operating System Design with Computer Network Communication Protocols", Proceedings 5th Data Communication Symposium, September 1977, pp. 4-19 - 4-25.
[evolution of operating systems, possible solutions to overhead and complexity, virtual terminal as end-to-end protocol]

Raphael, H.A., "Distributed Intelligence Microcomputer Design", Compcon 75 Spring Digest, 10th IEEE Computer Society International Conference, February 1975, pp. 21-26.
[microprocessors to provide I/O oriented functions, communication via connected common memory or interconnected I/O ports]

Reames, C.C. and M.T. Liu, "A Loop Network for Simultaneous Transmission of Variable-Length Messages", Proceedings 2nd Annual Symposium on Computer Architecture, January 1975, pp. 7-12.
[description of ring interface, automatic traffic regulation, comparison to Pierce and Newhall loops]

Richards, H. and A.E. Oldehoeft, "Hardware-Software Interactions in SYMBOL-2R's Operating System", Proceedings 2nd Annual Symposium on Computer Architecture, January 1975, pp. 113-118.
[major OS functions, virtual memory, high-level language compilation, timesharing implemented in hardware]

Ritchie, D.M. and K. Thompson, "The UNIX Timesharing System", Communications ACM, Volume 17, Number 7, July 1974, pp. 365-375.
[hierarchical OS, developed at Bell Labs, runs on PDP-11, compatible file, device and interprocess I/O using pipes, SHELL]

Roberts, L.G. and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing", Proceedings Spring Joint Computer Conference, Volume 36, May 1970, pp. 543-549.
[philosophy of networks, description of ARPA, IMPs, communication facilities, comparison and analysis]

Rosen, S., "Hardware Design Reflecting Software Requirements", Proceedings Fall Joint Computer Conference, Volume 33, November 1968, pp. 1443-1448.
[hardware advances alone cannot significantly improve total system effectiveness, importance of software and user interface]

Ruschitzka, M.G. and R.S. Fabry, "The PRIME Message System", Compcon 73 Digest of Papers, 7th IEEE Annual IEEE Computer Society International Conference, February 1973, pp. 125-128.
[medium timesharing system, highly modular architecture, network of 5 processors, 13 memory blocks, 15 disk drives, and 80 terminals, logical interconnection into virtual subsystems]

Sell, J.V., "Microprogramming in an Integrated Hardware/Software System", Computer Design, Volume 14, Number 1, January 1975, pp. 77-84.
[multiprogramming system, inexpensive, flexible, good performance from minicomputer]

Smith, W.R. et al., "SYMBOL - A Large Experimental System Exploring Major Hardware Replacement of Software",

Proceedings Spring Joint Computer Conference, Volume 38, May 1971, pp. 601-616.
[implementation of a high-level language, virtual memory, timesharing system that operates entirely without system software]

Sockhut, G.H., "Firmware/Hardware Support for Operating Systems: Principles and Selected History", SIGMICRO, Volume 6, Number 4, December 1975, pp. 17-26.
[criteria for which functions are best suited for implementation in firmware, good reference list]

Solomon, M.H. and R.A. Finkel, "The ROSCOE Distributed Operating System", Proceedings 7th Symposium on Operating System Principles, December 1979, pp. 108-114.
[operating system for a network of LSI-11s, no assumptions about topology of interconnection, concept of links, and messages for IPC]

Spier, M.J. and E.I. Organick, "The MULTICS Interprocess Communication Facility", Proceedings ACM 2nd Symposium on Operating System Principles, October 1969, pp. 83-91.
[exchange of messages among independent processes via mailbox structure, communication protocol]

Stockenberg, J. and A. van Dam, "Vertical Migration for Performance Enhancement in Layered Hardware/Software/ Software Systems", Computer, May 1978, pp. 35-50.
[automated methodology described, examination of static structure and dynamic behaviour of programs]

Stoy, J.E. and C. Strachey, "OS6 - An Experimental Operating System for a Small Computer", Computer Journal, 15, Number 2, 1972, pp. 117-124.
[single user system and reasons, list of requirements for an operating system]

Swan, R.J. et al., "Cm* - A Modular Multi-microprocessor", Proceedings AFIPS, Volume 46, 1977, pp. 637-644.

Titelbaum, M., "The LSI-11 - A System Microcomputer", Digest of Papers - EASCON 1975, Number 163, pp. 55-61.
[NMOS microprocessor with PDP-11 architecture and instruction set, microprogrammed to perform ASCII dialogue with operator]

Toong, H.M.D., "Microprocessors", Scientific American, September 1977, pp. 146-161.
[historical evolution of computers to ones on a single chip, importance of aids for program development]

Teichholtz, N.A., "Distributed Computing: A Modular Approach to Complex Systems", Compcon 75 Fall Digest, 11th IEEE Computer Conference, September 1975, pp. 137-138.
[distributed system of special function processors, makes analogy to human problem solving]

Tsichritzis, D.C. and P.A. Bernstein, Operating Systems, Academic Press, New York, 1974.
[principles governing the behaviour of operating systems, discusses process interaction, resource allocation, memory management, I/O, files, protection]

Walden, D.C., "A System for Interprocess Communication in a Resource Sharing Computer Network", Communications ACM, Volume 15, Number 4, April 1972, pp. 221-230.
[communication in timesharing system, extensions for distributed processes in a network, applied to ARPA]

Wecker, S., "A Design for a Multiple Processor Operating Environment", Compcon 73 Digest of Papers, 7th Annual IEEE Computer Society International Conference, February 1973, pp. 143-146.
[structure determines communication techniques, presents unified communication by explicit message exchange]

Weitzman, C., Minicomputer Systems: Structure, Implementation, and Application, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.
[excellent text on minicomputer architectures, peripherals, interfacing]

White, G.W., "Message Format Principles", ACM/IEEE 2nd Symposium on Problems in Optimization of Data Communications Systems, October 1971, pp. 192-198.
[definitive discussion of a generalized, standard message format based on ANSI Task group recommendations]

Whiting, J. and S. Newman, "Microprocessors in CRT Terminals", Proceedings National Computer Conference, Volume 44, May 1975, pp. 41-45.
[allows more complex functions, characteristics of a suitable micro, solutions to debugging: control panel, simulator, program debug]

Wulf, W.A. and C.G. Bell, "C.mmp - A Multi-Miniprocessor", Proceeding Fall Joint Computer Conference, Volume 41, Part II, December 1972, pp. 765-777.
[processors and memory connected by 16 point crossbar switch, design decisions in hardware and software]

MANUALS

_____, Model 980A Computer Assembly Language Coding Conventions, Texas Instruments Inc., March 1973.

_____, Model 980A Computer Assembly Language Input/Output, Texas Instruments Inc., April 1973.

_____, Model 980A Computer Basic System Use and Operation, Texas Instruments Inc., August 1972.

Ritchie, D.M., "C Reference Manual", Bell Telephone Labs, Murray Hill, New Jersey, pp. 1-30.

_____, MicroController Cross Assembly Program (MCCAP), Scientific Micro Systems, Mountain View, California.

_____, MicroController Digital System - System Description, Scientific Micro Systems, Mountain View, California.

_____, MicroController Digital System - Application Guide, Scientific Micro Systems, Mountain View, California.

APPENDICES

APPENDIX A - TI AND SMS BACKGROUND

APPENDIX B - HARDWARE DETAILS OF THE EXPERIMENT

APPENDIX C - MESSAGE FORMATS

APPENDIX D - DEVELOPMENT SYSTEM PROGRAMS

APPENDIX E - EXPERIMENTAL MMCMS SYSTEM PROGRAMS

APPENDIX A - TI AND SMS BACKGROUND

TI AND SMS INSTRUCTION SETS

TI980B Instruction Set

Memory Reference

ADD	Add to register A
AND	And with register A
BIX	Increment X register, branch if not zero
BRU	Branch unconditionally
CPA	Compare arithmetically
CPL	Compare logically
DAD	Double add to A & E registers
DIV	Divide A & E registers
DLD	Double load A & E registers
DMT	Decrement memory, skip next instruction if zero
DSB	Double subtract from A & E
DST	Double store A & E registers
IMO	Increment memory
IOR	Inclusive or with A register
LDA	Load A register
LDE	Load E register
LDM	Load M register
LDX	Load X register
MPY	Multiply E register, result in A & E registers
STA	Store A register
STE	Store E register
STX	Store X register
SUB	Subtract from A register

Register to Register

RAD	Add
RAN	And
RCA	Compare arithmetic
RCL	Compare logical
RCO	2's complement
RDE	decrement
REO	exclusive or
REX	exchange
RIN	increment
RIV	invert
RMO	move
ROR	inclusive or
RSU	subtract

Multiple Register

LRF	Load registers
SRF	Store registers
LSB	Load status block, PC & ST
SSB	Store status block
LSR	Load status block, reset

Shift

ALA	Arithmetic left A register
ALD	Arithmetic left double A & E registers
ARA	Arithmetic right A register
ARD	Arithmetic right double
CLD	Circular left double
CRA	Circular right A register
CRB	Circular right B register
CRD	Circular right double
CRE	Circular right E register
CRL	Circular right L register
CRM	Circular right M register
CRS	Circular right S register
CRX	Circular right X register
LLA	Logical left A register
LLD	Logical left double
LRA	Logical right A register
LRD	Logical right double
LTO	Left test for ones
LTZ	Left test for zeros
RTO	Right test for ones
RTZ	Right test for zeros

Skip Register

SEV	Skip even
SMI	Skip minus
SNO	Skip not all ones
SNZ	Skip not zero
SOD	Skip odd
SOO	Skip all ones
SPL	Skip plus
SZE	Skip zero

Skip Status Indicator

SEQ	Skip equal
SGE	Skip greater or equal
SGT	Skip greater
SLE	Skip less or equal
SLT	Skip less
SNC	Skip not carry
SNE	Skip not equal
SNV	Skip not overflow
SOC	Skip on carry
SOV	Skip on overflow

Skip Sense Switches

SSE	Skip switch equal
SSN	Skip switch not equal

Byte Manipulation

CLC Compare logical character
MVC Move character

Bit Manipulation - Memory

SMBO Set memory bit to one
SMBZ Set memory bit to zero
TMBO Skip next instruction if memory bit is one
TMBZ Skip if bit is zero

Bit Manipulation - A register

SABO Set bit to one
SABZ Set bit to zero
TABO Skip if bit is one
TABZ Skip if bit is zero

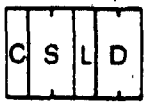
Input/Output

RDS Read 8 bits status, 8 bits data
WDS Write 8 bits control, 8 bits data
ATI DMA transfer initiate

Miscellaneous

API Auxiliary processor initiate
IDL Idles Pc
NRM Normalize

SMS300 MicroController Instruction Set

OPERATION	FORMAT	RESULT
MOVE		Content of data field addressed by S, L replaces data in field specified by D, L.
ADD		Sum of AUX and data specified by S, L replaces data in field specified by D, L.
AND		Logical AND of AUX and data specified by S, L replaces data in field specified by D, L.
XOR		Logical exclusive OR of AUX and data specified by S, L replaces data in field specified by D, L.
XMIT		The literal value I replaces the data in the field specified by S, L.
JMP		The literal value I replaces contents of the Program Counter.
NZT		If the data in the field specified by S, L equals zero, perform the next instruction in sequence. If the data specified by S, L is not equal to zero, execute the instruction at address determined by using the literal I as an offset to the Program Counter.
XEC		Perform the instruction at address determined by applying the sum of the literal I and the data specified by S, L as an offset to the Program Counter. If that instruction does not transfer control, the program sequence will continue from the XEC instruction location.

Comparison of Microcomputer Systems

Execution of Control-Oriented Benchmark Programs

Microcomputer System	Total Benchmark Execution Time (s)
SMS 300	39
Intel 8080	195
DEC PDP 8/A	246
National IMP-8	873
DEC MPS (Intel 8008)	3960

Source: from SMS in (Hodges, 1976)

```
                        PROG  TIMACRO
                       *LIBRARY OF MACROS TO EMULATE TI980 INSTRUCTION
                       *NOT ALL INSTRUCTIONS HAVE, OR CAN BE IMPLEMENTED
                       *SOME MNEMONICS OR FORMATS HAVE BEEN CHANGED
                       *ALL INSTRUCTIONS WORK ON 8 BITS OF DATA
                       *
                       *REGISTER EQUATES
000001                 A       EQU   R1
000002                 E       EQU   R2
000003                 X       EQU   R3
000004                 M       EQU   R4
000005                 S       EQU   R5
000011                 L       EQU   R11
000006                 B       EQU   R6
                       *
                       *BYTE AND BITS DEFINED
200 7 0                BYTWS   RIV   200H,7,8
200 0 1                BIT0    RIV   BYTWS,0,1
200 1 1                BIT1    RIV   BYTWS,1,1
200 2 1                BIT2    RIV   BYTWS,2,1
200 3 1                BIT3    RIV   BYTWS,3,1
200 4 1                BIT4    RIV   BYTWS,4,1
200 5 1                BIT5    RIV   BYTWS,5,1
200 6 1                BIT6    RIV   BYTWS,6,1
200 7 1                BIT7    RIV   BYTWS,7,1
                       *
                       *CONDITION CODE OF STATUS REGISTER
201 1 2                CC      RIV   201H,1,2
                       *DUMMY WORDS FOR ADDRESSING
200 7 0                BYTES1  RIV   BYTWS,7,8
200 7 0                BYTES2  RIV   BYTWS,7,8
                       *
                       RAD     MACRO SR,DR
                               MOVE  DR,AUX
                               ADD   SR,DR
                               ENDM
                       *
                       RAN     MACRO SR,DR
                               MOVE  DR,AUX
                               AND   SR,DR
                               ENDM
                       *
                       RCO     MACRO SR,DR
                               XMIT  -1,AUX
                               XOR   SR,DR      ASSUMES (SR)<200H, I.E. B
                               XMIT  1,AUX
                               ADD   DR,DR
                               ENDM
                       *
                       RDE     MACRO SR,DR
                               XMIT  -1,AUX
                               ADD   SR,DR
                               ENDM
                       *
                       REO     MACRO SR,DR
                               MOVE  DR,AUX
                               XOR   SR,DR
                               ENDM
```

```
                *
REX     MACRO SR,DR
        MOVE  SR,AUX
        MOVE  DR,SR
        MOVE  AUX,DR
        ENDM
*
RIN     MACRO SR,DR
        XMIT  1,AUX
        ADD   SR,DR
        ENDM
*
RINV    MACRO SR,DR
*THIS IS TT RIV INSTN, BUT CONFLICTS WITH RIGHT I
        XMIT  -1,AUX
        XOR   SR,DR
        ENDM
*
RMO     MACRO SR,DR
        MOVE  SR,DR
        ENDM
*
ROR     MACRO SR,DR
        MOVE  DR,AUX
        XOR   SR,DR
        AND   SR,AUX
        XOR   DR,DR
        ENDM
*
RSU     MACRO SR,DR
        XMIT  -1,AUX
        XOR   SR,DR
        ADD   DR,DR
        RIN   DR,DR
        ENDM
*
ADDO    MACRO ADR
        SEL   ADR
        RAD   ADR,A
        ENDM
*
ANDO    MACRO ADR
        SEL   ADR
        RAN   ADR,A
        ENDM
* *
BIX     MACRO ADR
        RIN   X,X
        NZT   X,ADR
        ENDM
*
BRL     MACRO NAME
        CALI  NAME
        ENDM
*
BRU     MACRO ADR
        JMP   ADR
        ENDM
```

```
117                 *
118         IMO     MACRO ADR
119                 SEL   ADR
120                 RIN   ADR,ADR
121                 ENDM
122                 *
123         IOR     MACRO ADR
124                 SEL   ADR
125                 ROR   ADR,A
126                 ENDM
127                 *
128         LDF     MACRO ADR
129                 SEL   ADR
130                 MOVF  ADR,E
131                 ENDM
132                 *
133         LDX     MACRO ADR
134                 SEL   ADR
135                 MOVF  ADR,X
136                 ENDM
137                 *
138         LDM     MACRO ADR
139                 SEL   ADR
140                 MOVF  ADR,M
141                 ENDM
142                 *
143         STA     MACRO ADR
144                 SEL   ADR
145                 MOVF  A,ADR
146                 ENDM
147                 *
148         STE     MACRO ADR
149                 SEL   ADR
150                 MOVF  E,ADR
151                 ENDM
152                 *
153         STX     MACRO ADR
154                 SEL   ADR
155                 MOVF  X,ADR
156                 ENDM
157                 *
158         SUB     MACRO ADR
159                 SEL   ADR
160                 RSU   ADR,A
161                 ENDM
162                 *
163         DLD     MACRO ADR
164                 SEL   ADR
165                 MOVF  ADR,A
166                 XMIT  ADR+1,IVR
167                 MOVF  BYTWS,E
168                 ENDM
169                 *
170         DST     MACRO ADR
171                 SEL   ADR
172                 MOVF  A,ADR
173                 XMIT  ADR+1,IVR
174                 MOVF  E,BYTWS
```

```
175                     ENDM
176                 *
177                 CRA MACRO C
178                     MOVE A(C),A
179                     ENDM
180                 *
181                 CRE MACRO C
182                     MOVE E(C),E
183                     ENDM
184                 *
185                 CRX MACRO C
186                     MOVE X(C),X
187                     ENDM
188                 *
189                 CRM MACRO C
190                     MOVE M(C),M
191                     ENDM
192                 *
193                 CRS MACRO C
194                     MOVE S(C),S
195                     ENDM
196                 *
197                 CRL MACRO C
198                     MOVE L(C),L
199                     ENDM
200                 *
201                 CRB MACRO C
202                     MOVE B(C),B
203                     ENDM
204                 *
205                 MVC MACRO
206                 MVC1 MOVE A,IVR
207                     MOVE BYTES1,AUX
208                     MOVE E,IVR
209                     MOVE AUX,BYTES2
210                     RIN A,A
211                     ADD E,E         INCR E ALSO
212                     RDE X,X
213                     NZT X,MVC1
214                     ENDM
215                 *
216                 CLC MACRO
217                     NZT X,CLC1
218                     JMP ALLMATCH
219                 CLC1 MOVE A,IVR
220                     MOVE BYTES1,AUX
221                     MOVE F,IVR
222                     XOR BYTES2,AUX
223                     NZT AUX,NOMATCH
224                     RIN A,A
225                     ADD F,E         INCR E ALSO
226                     RDE X,X
227                     NZT X,CLC1
228                 ALLMATCH SFL CC
229                     XMIT 01B,CC
230                     JMP ECLC
231                 NOMATCH XMIT -1,AUX
232                     XOR BYTES2,AUX
```

```
233                     SEL   BYTWS
234                     MOVF  AUX,BYTWS
235                     XMIT  1,AUX
236                     ADD   BYTWS,AUX
237                     MOVF  A,IVR
238                     ADD   BYTES1,AUX
239                     SEL   BYTWS
240                     MOVF  AUX,BYTWS
241                     NZT   BIT0,CLC2
242                     SEL   CC
243                     XMIT  10B,CC      BYTES1>BYTES2
244                     JMP   ECLC
245           CLC2      SEL   CC
246                     XMIT  00B,CC      BYTES1<BYTES2
247           ECLC      ENDM
248           *
249           LDA       MACRO ADR
250                     XMIT  ADR,IVR
251                     MOVF  ADR,A
252                     ENDM
253           *
254           LDAB      MACRO DISP
255                     XMIT  DISP,AUX
256                     ADD   B,IVR
257                     MOVF  BYTWS,A
258                     ENDM
259           *
260           LDAX      MACRO ADR
261                     XMIT  ADR,AUX
262                     ADD   X,IVR
263                     MOVF  BYTWS,A
264                     ENDM
265           *
266           LDABX     MACRO DISP
267                     XMIT  DISP,AUX
268                     ADD   B,AUX
269                     ADD   X,IVR
270                     MOVF  BYTWS,A
271                     ENDM
272           *
273           LDAI      MACRO ADR
274                     XMIT  ADR,IVR
275                     MOVF  ADR,IVR
276                     MOVF  BYTWS,A
277                     ENDM
278           *
279           LDABI     MACRO DISP
280                     XMIT  DISP,AUX
281                     ADD   B,IVR
282                     MOVF  BYTWS,IVR
283                     MOVF  BYTWS,A
284                     ENDM
285           *
286           LDAIX     MACRO ADR
287                     XMIT  ADR,A
288                     MOVF  ADR,AUX
289                     ADD   X,IVR
290                     MOVF  BYTWS,IVR
```

```
                        ENDM
                *
LDAXI   MACRO ADR
        XMIT  ADR,AUX
        ADD   X,IVR
        MOVF  BYTWS,IVR
        MOVF  BYTWS,A
        ENDM
                *
LDAIM   MACRO DISP
        XMIT  DISP,A
        ENDM
                *
LDMR    MACRO DISP
        XMIT  DISP,AUX
        ADD   B,IVR
        MOVF  BYTWS,M
        ENDM
                *
LDEIM   MACRO DISP
        XMIT  DISP,E
        ENDM
                *
LDXIM   MACRO DISP
        XMIT  DISP,X
        ENDM
                *
LDMIM   MACRO DISP
        XMIT  DISP,M
        ENDM
                *
STAX    MACRO ADR
        XMIT  ADR,AUX
        ADD   X,IVR
        MOVF  A,BYTWS
        ENDM
                *
ADDIM MACRO DISP
      XMIT  DISP,AUX
      ADD   A,A
      ENDM
                *
ANDIM MACRO DISP
      XMIT  DISP,AUX
      AND   A,A
      ENDM
                *
SARZ    MACRO BIT
        XMIT  376H,AUX
        MOVF  AUX(BIT+1),AUX
        AND   A,A
        ENDM
                *
SARO    MACRO BIT
        SFL   BYTWS
        MOVF  A,BYTWS
        XMIT  1,30H+BIT
        MOVF  BYTWS,A
```

```
                        ENDM
                *
                *
TAR0    MACRO BIT,ADR
        XMIT  1,AUX
        MOVE  AUX(BIT+1),AUX
        AND   A,AUX
        NZT   AUX,ADR
        ENDM
                *
TAR7    MACRO BIT,ADR
        XMIT  1,AUX
        MOVE  AUX(BIT+1),AUX
        AND   A,AUX
        NZT   AUX,*+2
        JMP   ADR
        ENDM
                *
SMR0    MACRO BIT,ADR
        SEL   ADR
        XMIT  1,30H+BIT
        ENDM
                *
TMR0    MACRO BIT,ADR,JADR
        SEL   ADR
        NZT   30H+BIT,JADR
        ENDM
                *
LRA     MACRO C
P       SET   8-C
        XMIT  1,L.P-1,AUX
        AND   A(C),A
        ENDM
                *
DMT     MACRO MLOC,ADR
        SEL   MLOC
        RDE   MLOC,MLOC
        NZT   MLOC,*+2
        JMP   ADR
        ENDM
                *
*ALL BXX INSTRUCTIONS WILL OPERATE ON REGISTERS O
*WORKING STORAGE IF SELECTED BEFOREHAND
BNE     MACRO R,CHR,ADR
*GO TO ADR IF (R).NE.CHR
        XMIT  CHR,AUX
        XOR   R,AUX
        NZT   AUX,ADR
        ENDM
                *
BEQ     MACRO R,CHR,ADR
*GO TO ADR IF (R)=CHR
        XMIT  CHR,AUX
        XOR   R,AUX
        NZT   AUX,*+2
        JMP   ADR
        ENDM
                *
```

```
                BLT     MACRO R,NO,ADR
                        XMIT  -NO,AUX
                        ADD   R,AUX
                        SFL   BYTWS
                        MOVE  AUX,BYTWS
                        NZT   BIT0,ADR
                        ENDM
                *
                BZF     MACRO D,ADR
                        NZT   D,*+2
                        JMP   ADR
                        ENDM
                *
                BNZ     MACRO D,ADR
                        NZT   D,ADR
                        ENDM
                *
                BPL     MACRO R,ADR
                        XMIT  200H,AUX
                        AND   R,AUX
                        NZT   AUX,ADR
                        ENDM
                *
                BMI     MACRO R,ADR
                        XMIT  200H,AUX
                        AND   R,AUX
                        NZT   AUX,*+2
                        JMP   ADR
                        ENDM
                *
                BOD     MACRO R,ADR
                        XMIT  1,AUX
                        AND   R,AUX
                        NZT   AUX,*+2
                        JMP   ADR
                        ENDM
                *
                BNO     MACRO R,ADR
                        XMIT  1,AUX
                        ADD   R,AUX
                        NZT   AUX,ADR
                        ENDM
                *
                BOO     MACRO R,ADR
                        XMIT  1,AUX
                        ADD   R,AUX
                        NZT   AUX,*+2
                        JMP   ADR
                        ENDM
                *
                REV     MACRO R,ADR
                        XMIT  1,AUX
                        AND   R,AUX
                        NZT   AUX,ADR
                        ENDM
                *
                DATA    MACRO NAME,VAL
                        SEL   NAME
```

```
465                     XMIT    VAL,AUX
466                     MOVF    AUX,NAME
467                     ENDM
```

```
                        *SAMPLE TI ASSEMBLER PROGRAM
                        *
          202 7 0       MEMLOC  RIV     202H,7,8
          203 7 0       OFS     RIV     203H,7,8
          204 7 0       NOLIN   RIV     204H,7,8
          205 7 0       OUTBUF  RIV     205H,7,8
          206 7 0       PRCHR8  RIV     206H,7,8
          207 7 0       PRBL    RIV     207H,7,8
          210 7 0       PRBC    RIV     210H,7,8
          211 7 0       SAVS    RIV     211H,7,8
          212 7 0       SAVX    RIV     212H,7,8
          213 7 0       ASCZ1   RIV     213H,7,8
                        *
                                DATA    ASCZ1,'0'
                                DATA    PRBL,2
                        *
                        DUMP    LDAR    1
                                STA     MEMLOC
                                LDAR    3
                                STA     OFS
                                LDAR    0
                                LRA     3
                                BZE     A,NI1
                                BPL     A,NI2
                        NI1     LDATM   1
                        NI2     STA     NOLIN
                        *
                        *ALTERNATIVELY USE A REGISTER FOR NOLIN
00035  6 04001                  XMIT    1,R4        DEFAULT # LINES OF DUMP
                                LDMR    2
                        NEWLN   RMO     M,S
                                LDEIM   OUTBUF
                                RMO     E,B
                                LDXIM   -9
                                BRU     CON
                        DMP2    LDAI    MEMLOC
                                RMO     A,M
                                IMO     MEMLOC
                                BRL     STPRNT
                                BRL     STPRNT
                                STAX    PRCHR8
                        CON     BRL     BINHEX
                        *DSTR 0
                                LDATM   3
                                RAD     A,B
                                BIX     DMP2
                                LDMIM   PRBL
00075  6 11003                  CALL    TORHAN
00076  7 00141
                        *
                        *CHECK THE ERROR AND IGNORE BITS IN PRB STATUS
                                LDA     PRBL
                                TABO    1,XIT
                                TABO    3,XIT
                        *
                        *ALTERNATIVELY, USE THE FOLLOWING TO CHECK STATUS
                                TMBO    1,PRBL,XIT
00113  5 33134                  NZT     BIT3,XIT
```

```
 526  00114  6 17203          SFL    OFS          SMS CODE
 527  00115  4 37017          XFC    INSTN(OFS)   SMS CODE
 528                           RRU    *+3
 529                    INSTN  LDMTM  8
 530                           LDMTM  16
 531                           RAD    S,M
 532                           DMT    NOLIN,XIT
 533                           RRU    NEWLN
 534                    *
 535                    *ALTERNATIVELY, USING R4 FOR NOLIN, CHECK FOR MOR
 536                           RDE    R4,R4
 537                           BNZ    R4,NEWLN
 538                    XIT    LDMTM  PRRC
 539  00135  6 11004          CALI   IORHAN
      00136  7 00141
 540                           RRU    *
 541                    *
 542  00140                    PROC   STPRNT       NOT IMPLEMENTED
 543  00140  7 00210          RTN
 544                           END    STPRNT
 545                    *
 546  00141                    PROC   IORHAN       NOT IMPLEMENTED
 547  00141  7 00210          RTN
 548                           END    IORHAN
 549                    *
 550  00142                    PROC   BINHEX
 551                           STX    SAVX
 552                           RMO    S,X
 553                           STX    SAVS
 554                           DLD    ASCZ1
 555                           LDXIM  -4
 556                    MERGLP RMO    M,S
 557                           REX    A,S
 558                           ANDIM  17H
 559                           BLT    A,10,NI3
 560                    NI3    ADDIM  7
 561                           REX    A,S
 562                           RAD    S,E
 563                    *CRD 8
 564                           CRM    4
 565                           BIX    MERGLP
 566                           LDX    SAVS
 567                           RMO    X,S
 568                           LDX    SAVX
 569  00207  7 00210          RTN
 570                           END    BINHEX
 571                           END    TIMACRO


RETURN TABLE

      00210  4 11211
      00211  7 00057
      00212  7 00061
      00213  7 00066
      00214  7 00077
      00215  7 00137
```

APPENDIX B - HARDWARE DETAILS OF THE EXPERIMENT

B.1 CONTRAST BETWEEN BUS AND MESSAGE COMMUNICATION

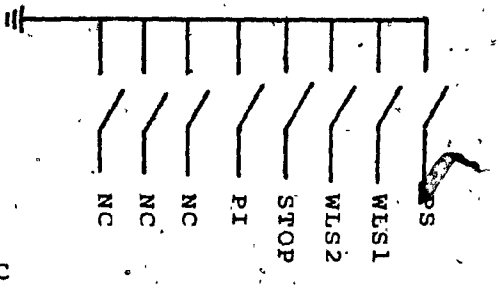
Bus communication is a means of transmitting a data item from a device which generates an address associated with the data item to a device which uses the address to recognize data items directed to it. The bus is effectively the medium of communication.

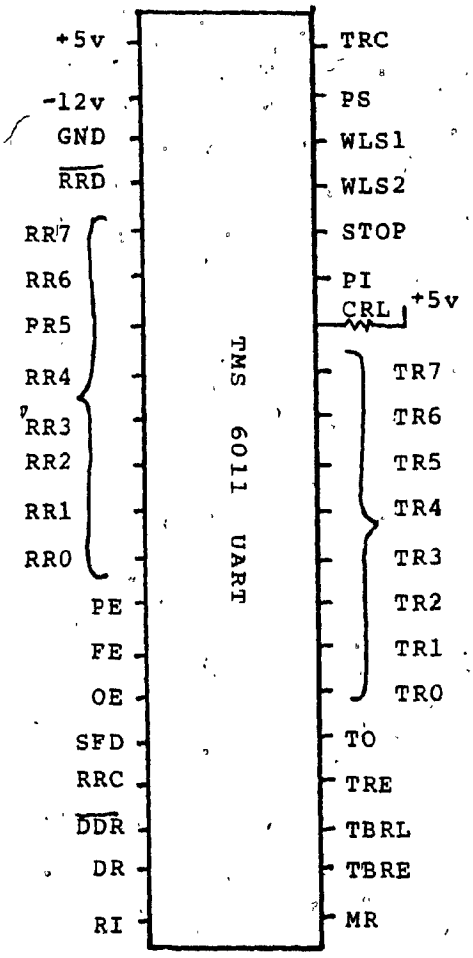
A message is an organization of a set of data items consisting of a header, text and end character. A message may be transmitted on a variety of physical media. In the case of information transfer between the SMS and the TI the data is organized as messages addressed by name to the device or routine. The format is outlined in Section 3.3.2. A routine called COMDSR is responsible for the transmission and reception of messages sent on the bus. The TI I/O bus and SMS Interface Vector are connected through a shared memory.

B.2 VDU INTERFACE TO MICROPROCESSOR

The VDU is interfaced to the microprocessor through a standard Texas Instruments TMS 6011 Universal Asynchronous Receive/Transmit (UART) integrated circuit. Data is loaded in 8 bits parallel into the transmit buffer register through interface vector byte 5. Load synchronism is achieved through inspection of the Transmit Buffer Register Empty (TBRE) status through IV byte 4. Similarly, the received information is read from the Receive Buffer Register (RRO-RR7) in 8 bits parallel through IV byte 4. The interface transmission speed is controlled by the switch selected baud rate implemented with a dual in line switch and a 1441 Baud Rate Generator. Word format is selected by a second dual in line switch which connects the appropriate 6011 pins to ground. Other status and control bits as shown in the schematic are monitored or controlled through IV bytes 3 and 4.

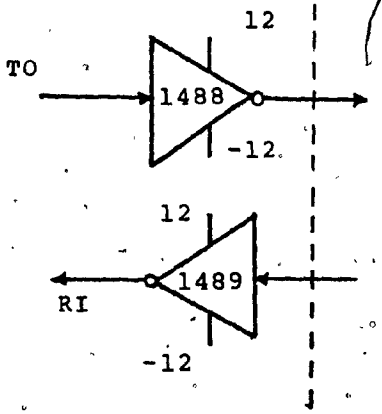
VDU INTERFACE TO MICROPROCESSOR

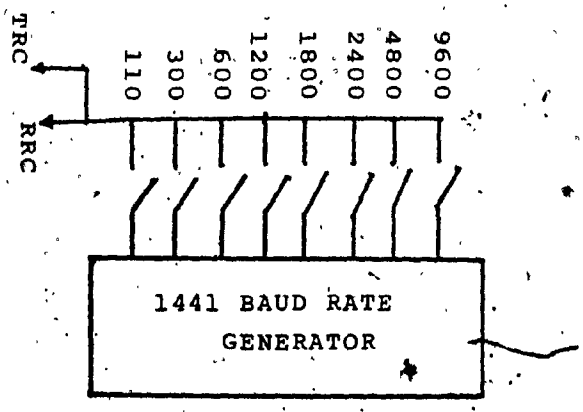


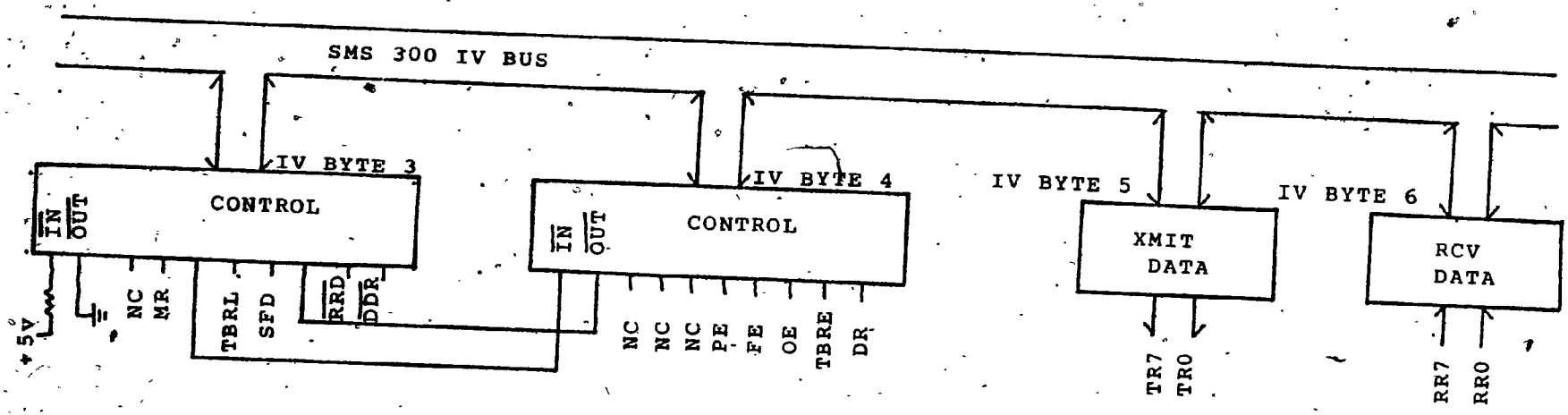


TRANSMIT DATA REGISTER

RS-232C INTERFACE FOR VDU







SMS Interface to UART

VDU INTERFACE TO MICROPROCESSOR

List of Mnemonics

TRC	Transmitter Register Clock
PS	Parity Select
WLS1	Word Length Select
WLS2	Word Length Select
STOP	Stop Bit Select
PI	Parity Inhibit
CRL	Control Register Load
TR0-TR7	Transmit Register Parallel inputs
TO	Serial Output
TRE	Transmit Register Empty, Status bit
TBRL	Transmit Buffer Register Load
TBRE	Transmit Buffer Register Empty
MR	Master Reset
RRD	Receive Register Disable
RR0-RR7	Receive Register Parallel outputs
PE	Parity Error
FE	Framing Error
OE	Overflow Error
SFD	Status Flag Disable
RRC	Receive Register Clock
DDR	Data Receive Reset
DR	Data Ready
RI	Serial Input

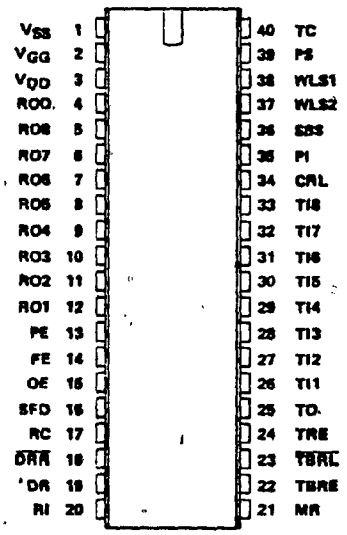
MOS
LSI

TMS 6011 JC, NC
ASYNCHRONOUS DATA INTERFACE (UART)

BULLETIN NO. DL-S 7512275, REVISED NOVEMBER 1977

- Transmits, Receives, and Formats Data
- Full-Duplex or Half-Duplex Operation
- Operation from DC to 200 kHz
- Static Logic
- Buffered Parallel Inputs and Outputs
- Programmable Word Lengths . . . 5, 6, 7, 8 Bits
- Programmable Information Rate
- Programmable Parity Generation/Verification
- Programmable Parity Inhibit
- Automatic Data Formatting
- Automatic Status Generation
- 3-State Push-Pull Buffers
- Low-Threshold Technology
- Standard Power Supplies . . . 5 V, −12 V
- Full TTL Compatibility . . . No External Components

40-PIN CERAMIC AND PLASTIC
DUAL-IN-LINE PACKAGES
(TOP VIEW)



description

The TMS 6011 JC, NC is an MOS/LSI subsystem designed to provide the data interface between a serial communications link and data processing equipment such as a peripheral or a computer. The device is often referred to as an asynchronous data interface or as a universal asynchronous receiver/transmitter (UART).

The receiver section of the TMS 6011 will accept serial data from the transmission line and convert it to parallel data. The serial word will have start, data, and stop bits. Parity may be generated and verified. The receiver section will validate the received data transmission by checking proper start, parity, and stop bits, and will convert the data to parallel.

The transmitter section will accept parallel data, convert it to serial form, and generate the start, parity, and stop bits.

The TMS 6011 is a fully programmable circuit allowing maximum flexibility of operation, defined as follows:

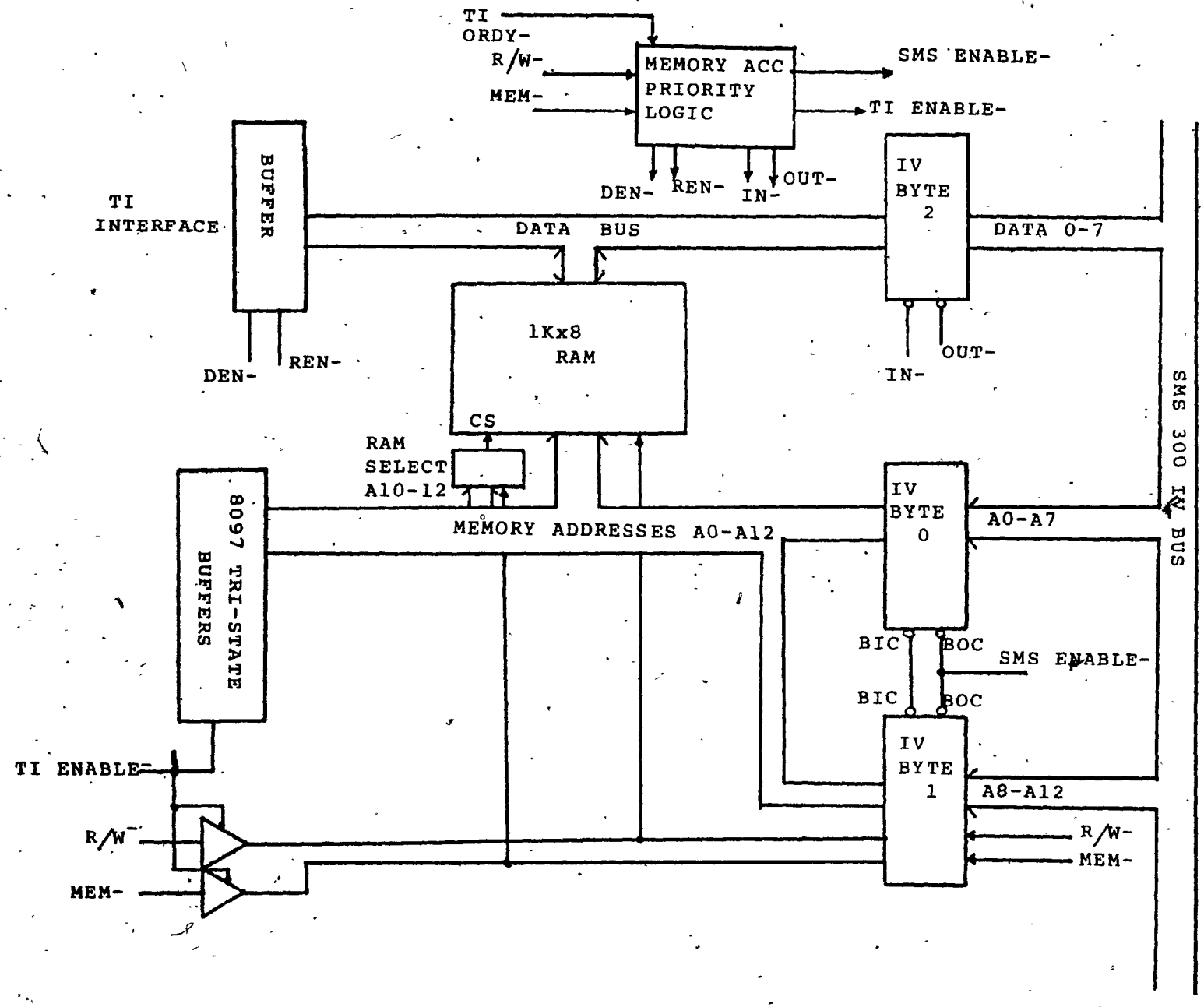
- The receiver and transmitter sections are separate and can operate either in full-duplex (simultaneous transmission and reception) or in half-duplex mode (alternate transmission and reception).
- The data word may be externally selected to be 5, 6, 7, or 8 bits long.
- Baud rate is externally selected by the clock frequency. Clock frequency can vary between 0 and 200 kHz.
- Parity, which is generated in the transmit mode and verified in the receive mode, can be selected as either odd or even. It is also possible to disable the parity bit by inhibiting the parity generation and verification.
- The stop bit can be selected as either a single- or a double-bit stop.
- Static logic is used to maximize flexibility of operation and to simplify the task of the user. The data holding registers are static and will hold a data word until it is replaced by another word.
- Asynchronous operation allows the use of a single transmission line. The clock period has to be within ±4% of 1/16 of the time for one bit for the transmitter and/or receiver but no phase relationship is required.

To allow for a wide range of possible configurations, three-state push-pull buffers have been used on all outputs except Transmitter Output (TO) and Transmitter Register Empty (TRE). They allow the wire-OR configuration.

UART

B.3 SMS DATA MEMORY OPERATION

The SMS Program Storage is accessible through three SMS IV bytes. The memory address is transmitted through IV bytes 0 and 1, while the data is received or transmitted through IV byte 2. Read or Write is controlled by 2 bits of IV byte 1, which in turn are fed to the priority logic to control the direction of IV byte 2. The TI 980B minicomputer also has access to this memory through an interface consisting of a 8097 tri-state buffer. The Memory Access Priority Logic gives absolute priority to requests from the TI. It also generates all control signals required to manage the interfaces of the two processors.


TI
ORDY-
R/W-
MEM-
MEMORY ACC
PRIORITY
LOGIC
SMS ENABLE-
TI ENABLE-
DEN-
REN-
IN-
OUT-
TI
INTERFACE
BUFFER
DATA BUS
IV
BYTE
2
DATA 0-7
1Kx8
RAM
DEN-
REN-
IN-
OUT-
CS
RAM
SELECT
A10-12
MEMORY ADDRESSES A0-A12
8097 TRI-STATE
BUFFERS
IV
BYTE
0
A0-A7
SMS 300 IV BUS
BIC
BOC
SMS ENABLE-
BIC
BOC
IV
BYTE
1
A8-A12
TI ENABLE-
R/W-
MEM-
R/W-
MEM-

B.4 OPERATOR CONSOLE COMMANDS FOR TI980

In the operator-computer dialogues described in Chapter 3, the following symbols are used:
The symbol) is the return key signifying the end of the keyin.
Characters within square brackets [ ] are optional. A command keyword is a maximum of six characters but only as many as are necessary to make it unique are required.
Words in parentheses ( ) are for explanation or comment and are not keyed in or printed.
The following list is a summary of operator console commands:

COMMAND	USE
A[SSIGN],lun,pd,filnam	peripheral selection
CA[LL],filnam	executes a procedure
DEF[INE],disc,filnam,recsiz,type,extent	disc file creation
DEL[ETE],disc,filnam	disc file deletion
DM[PMEM],mem,fadr,ladr	dump memory
[EXECUT,][pd,]filnam	load/execute
LO[AD],[pd,]filnam	load a program
LU[NOS]	logical units assigned.
M[MAP]	map of memory
REL[EAS],lun	peripheral selection
RET[URN]	returns from a procedure
REW[IND],lun	positioning

ABBREVIATION	MEANING
lun	logical unit number (base 16)
pd	physical device names, e.g. LOG, CR
filnam	disc file name

ABBREVIATION	MEANING
disc	D0 or D1
recsiz	number of characters in a disc record
type	blocked or unblocked, temporary or permanent
extent	disc file size in physical records
mem	memory type e.g. TI, SMS, PS
fadr	first address of memory to be dumped
ladr	last address

APPENDIX C - MESSAGE FORMATS

Distributed Computing System (Farber, 1972a), (Farber, 1972b)

1) Originating physical RI number, parity checked (9 bits)
2) Ring check (1 bit)
3) Terminating Process Name/TPN (16 bits)
4) Originating Process Name/OPN (16 bits)
5) Header check bit parity (1 bit)
6) Serial Field (1 bit)
7) Message Definition Field/MDF (8 bits)
8) Text (about 1000 bits)
9) Matched Bit/MB (1 bit)
10) Accepted Bit/AB (1 bit)
11) Whole message check, N bit error detecting

Newhall Loop (Farmer, 1969)

1) SOM/Start of Message (6 bits)
2) Destination (6 bits)
3) Originator (6 bits)
4) Opcode (4 bits)
5) Data (variable length)
6) EOM/End of Message (6 bits)

National Security Agency (Hassing, 1973)

1) NIP header (40 bits)
   - Originator Address (16 bits)
   - Destination Address (16 bits)
   - Flag (4 bits)
   - Tally (4 bits)
2) Header CRC (16 bits)
3) CHIP header (48 bits)
   - Control Link (8 bits)
   - Function (8 bits)
   - Data Link (8 bits)
   - Message Identification (8 bits)
   - Displacement (16 bits)
4) Control (8 bits)
5) Text (640 bits)
6) Packet CRC (16 bits)

Distributed Loop Computer Network (Reames, 1975), (Liu, 1977)

1) Flag (8 bits)
2) Destination Address (12 bits)
   - Loop interface address (7 bits)
   - Process number (5 bits)
3) Origin Address (12 bits)
4) Message Control Field (16 bits)
   - Message type (2 bits)
   - Broadcast (1 bit)
   - Function (3 bits)
   - Lost message detection (2 bits)
   - Lock-out prevention (1 bit)
5) Information (variable length)
6) CRC (16 bits)
7) Flag (8 bits)

Aloha (Abramson, 1973)

1) Terminal number
2) Record length — Header (32 bits)
3) CRC code for header (16 bits)
4) Text (320 or 640 bits)
5) CRC on entire message (16 bits)

ARPA Network (Heart, 1970)

SYN

Hardware generated

SYN

1) DLE STX Start frame
2) Header:
   - ACK (1 bit)
   - For IMP (1 bit)
   - Trace (1 bit)
   - RFNM (1 bit)
   - Priority (1 bit)
   - Discard (1 bit)
   - Spares (9 bits)
   - Line test (1 bit)
   - Last packet (1 bit)
   - Message number (7 bits)
   - Destination (8 bits)
   - Conversion (1 bit)
   - From IMP (1 bit)
   - Zeros (6 bits)
   - Source (8 bits)
   - Link number (8 bits)
   - Spares
   - Packet number

3) Text (1000 bits per packet)
(8095 bits per host message)
4) DLE ETX End frame
5) Check characters - error control Hardware generated

ANSI X3 Project 281 (Emmons, 1977)

1) Flag (8 bits)
2) Address (8 bits)
3) Control (8 bits)
4) Network Control Heading (variable)
Heading Item
HIC/Heading Item Code (4 bits)
Length (4 bits)
Parameter (variable)

HIC (4 bits)
Parameter (4 bits)

HIC (4 bits)
Extender to HIC (4 bits)
Length (8 bits)
Parameter (variable)
5) System/User Control Headings
6) Data
7) Frame check sequence (16 bits)
8) Flag (8 bits)

APPENDIX D - DEVELOPMENT SYSTEM PROGRAMS

```
                    0001          IDT   JCDMPM
                    0002  *DMPMEM,TYPE,LOWER,UPPER.  OUTPUTS TO LUN >20
                    0003          DEF   JCDMPM
                    0004          REF   DMPMEM
                    0005          REF   JCASHX
                    0006          REF   JCSTRT
                    0007          REF   MSGSM
                    0008          REF   GETBUF,PUTBUF
                    0009          REF   OTADR
                    0010          REF   INP
                    0011          REF   INBYTE
                    0012  *
                    0013  *REGISTER AND OTHER EQUATES
              0000  0014  A       EQU   0
              0001  0015  E       EQU   1
              0002  0016  X       EQU   2
              0003  0017  M       EQU   3
              0004  0018  S       EQU   4
              0005  0019  L       EQU   5
              0006  0020  B       EQU   6
              0007  0021  PC      EQU   7
              0001  0022  BR      EQU   1
   0000       C4CD  0023  JCDMPM  DATA  'DMPMEM'
P  0003       0007  0024        DATA  START
P  0004       0085  0025        DATA  LASTLN
                    0026  *
              0002  0027  PSTRT   DATA  JCSTRT
X  0005       0000
              0001  0028  PASHX   DATA  JCASHX
X  0006       0000
                    0029  *
              0007  0030  START   EQU   $
                    0031  *ON ENTRY A-REG CONTAINS ADR OF FST
                    0032  *FST: # PARMS, 3 WORDS ASCII, 3 WORDS ASCII, ...
                    0033  *PARMS: # WORDS TO DUMP, MEMORY LOCN TO DUMP FROM,
                    0034  *       STARTING ADR FOR LISTING, OFFSET FOR NEW LINE ADR
   0007       C506  0035        RMO   A,B       SAVE FST ADR
   0008       2707  0036        ADD   =7
   0009       74FC  0037        BRL   *PASHX    GET 2ND OPERAND, LOWER ADDR
```



```
   000A       8028  0038        STA   PARMS+2   LOWER IS STARTING ADR
   000B       C5C0  0039        RMO   B,A       GET BACK FST ADR
   000C       270A  0040        ADD   =10
   000D       74F8  0041        BRL   *PASHX      GET 3RD OPERAND, UPPER ADDR
   000E       2824  0042        SUB   PARMS+2   UPPER-LOWER
   000F       CCE0  0043        SPL   A
   0010       0700  0044        LDA   =0        EVEN IF INCORRECT DUMP 1 LINE
   0011       2708  0045        ADD   =8        +1 & CEILING, # WORDS TO DUMP
   0012       801E  0046        STA   PARMS
   0013       3104  0047        LDA   4,BR      GET MEMORY TYPE AS KEYED IN
   0014       17FC  0048        LDX   =-4       -VE # TYPES FOR BIX
   0015       620E  0049        CPL   MTYPE+4,X   CHECK LIST OF MEMORY TYPES
   0016       CDA0  0050        SNE             DO THEY MATCH?
   0017       7A21  0051        BRU   MATCH,X   YES
   0018       40FC  0052        BIX   $-3       NO, CONTINUE CHECKING
   0019       8011  0053        STA   MTP       TYPE NOT FOUND SO PLUG ERROR MSG
   001A       1000  0054        @LDX  =ERMS1
   001C       DBC0  0055        @SSB  MSGSM
P  001E       0000  0056        DATA  JCDMPM
   001F       7CE5  0057        BRU   *PSTRT
                    0058  *
   0020       D4C9  0059  MTYPE  DATA  'TISYPSMS'
   0024       000C  0060  ERMS1  DATA  12,'MEMORY TYPE '
   0028       A0A0  0061  MTP    DATA  '  ',' NOT FOUND'
P  0031            0062  PARMS  BSS   4
                    0063  *
              0034  0064  MATCH  EQU   $+4
   0035       7803  0065        BRU   TI
   0036       7831  0066        BRU   SM
   0037       7808  0067        BRU   PS
   0038       7800  0068        BRU   MS
                    0069  *
              0039  0070  MS     EQU   $
              0039  0071  TI     EQU   $
   0039       00F9  0072        LDA   PARMS+2   STARTING DUMP ADR
   003A       80F7  0073        STA   PARMS+1
   003B       0000  0074        LDA   $+1       GET INSTN TO INCR ADR BY 8
   003C       1F06  0075        LDX   =6
   003D       10F5  0076        STX   PARMS+3   PASS IT TO DMPMEM
```

```
003E  1800  0077          @LDM  =PARMS      ; REG PTS TO PARMS
0040  D8C0  0078          @SSB  DMPMEM      GO DUMP MEMORY
0042  7CC2  007Y          BRU   *PSTRT
            0080  *
      0043  0081  PS      EQU   $
0043  00EC  0082          LDA   PARMS       # WORDS TO DUMP
0044  *7400  0083          @BRL  GETBUF      GET TI MEMORY FOR PS
0046  63EA  0084          CPA   PARMS       RETURNS # WORDS IN A-REG
0047  CDC0  0085          SLE
0048  CE01  0086          IDL   1           MEMORY REQUESTED WAS NOT AVAILABLE
0049  C102  0087          RCO   A,X         -VE FOR BIX
004A  38E7  0088          SIE   PARMS+1     TI ADR TO DUMP FROM
004B  C516  0089          RMO   E,B         THEREFORE, ADR TO READ PS MEMORY INTO
004C  30E6  0090          LDA   PARMS+2     STARTING ADR FOR DUMP
004D  2000  0091          @ADD  =>4000      SET ADR FOR LSB 2ND BYTE
004F  D8C0  0092          @SSB  OTADR       SET ADR ON INTERFACE
            0093  *READ IN ALL LOW BYTES FIRST
0051  7400  0094  FBYTE   @BRL  INBYTE      GET 1 BYTE
0053  8100  0095          STA   0,BR
0054  C366  0096          RIN   B,B         PT TO NEXT WORD
0055  40F8  0097          BIX   FBYTE       LOOP UNTIL DONE
0057  30DC  0098          LDA   PARMS+2     RETRIEVE STARTING ADR
0057  2000  0099          @ADD  =>2000      RESET ADR TO 1ST BYTES
0059  D8C0  0100          @SSB  OTADR
005B  B0D5  0101          DLD   PARMS       GET COUNT & ADR
005C  C102  0102          RCO   A,X
005D  C516  0103          RMO   E,B
            0104  *READ IN MSB BYTES NOW, COMBINE WITH ONES ALREADY THERE
005E  7400  010Y  NBYTE   @BRL  INBYTE      GET 1 BYTE
0060  C8C8  0106          LLA   8           SHIFT IT UP
0061  3100  0107          IOR   0,BR        COMBINE WITH OTHER BYTE
0062  8100  0108          STA   0,BR
0063  C366  010Y          RIN   B,B         PT TO NEXT
0064  40F9  0110          BIX   NBYTE       LOOP UNTIL DONE
0065  0000  0111          LDA   $+1         GET INSTN THAT INCR ADR BY 8
0066  1E08  0112          LDM   =8
0067  7814  0113          BRU   SMSMEM
            0114  *
      0068  0115  SM      EQU   $
```



```
0068  00C8  0116          LDA   PARMS       GET # WORDS TO DUMP
0069  2708  0117          ADD   =8          ADD IN ANOTHER 8 SINCE SM ADRS GO BY 16
006A  C801  0118          ARA   1           # WORDS TO GET & DUMP
006B  80C5  0119          STA   PARMS
006C  7400  0120          @BRL  GETBUF      GET TI MEMORY FOR S4 LOCNS
006E  98C2  0121          CPA   PARMS       DID WE GET WHAT WE ASKED FOR?
006F  CDC0  0122          SLE
0070  CE01  0123          IDL   1           MEMORY REQUESTED WAS NOT AVAILABLE
0071  C881  0124          ALA   1           X2 TO GET # CHARS
0072  C102  0125          RCO   A,X                 -VE FOR BIX COUNTER
0073  88BE  0126          SIE   PARMS+1     TI ADR TO DUMP FROM
0074  C516  0127          RMO   E,B         ADR TO READ SM MEMORY INTO
0075  00BD  0128          LDA   PARMS+2     GET STARTING ADR FOR DUMP
0076  D8C0  012Y          @SSB  OTADR
            0130  *X=-VE # CHARS TO READ (DUMP); B=BUFFER ADR
0078  D8C0  0131          @SSB  INP         INPUT FROM SMS MEMORY
007A  0000  0132          LDA   $+1         GET INSTN TO INCR ADR BY 16
0078  1F10  0133          LDM   =16
007C  8087  0134  SMSMEM  STA   PARMS+3     PASS IT TO DMPMEM
007D  1800  0135          @LDM  =PARMS      ; REG PTS TO PARMS
007F  D8C0  0136          @SSB  DMPMEM
0081  B0AF  0137          DLD   PARMS       GET WORD COUNT & BUFR ADR TO GIVE BACK
0082  7400  0138          @BRL  PUTBUF
0084  7C80  013Y          BRU   *PSTRT
            0140  *
      0085  0141  LASTLN  EQU   $
      0000  0142          END
```

DIGITAL SYSTEMS LABORATORY
COMPUTER SCIENCE DEPARTMENT
CONCORDIA UNIVERSITY - MONTREAL QUEBEC

PROGRAM MNEMONIC: JCDMPM | REVISION LEVEL: R2.0/V** | REV DAT: 27

- JCDMPM - MEMORY DUMP IN HEX & ASCII

PAGE

	A	0000	B	0006	BR	0001	DMPMEM	0000
	E	0001	ERRST	0024	FBYTE	0051	GETBUF	0004
	INBYTE	0025	INF	0027	JCASHX	0001	JCDMPM	0000
	JCSTAT	0002	L	0005	LASTLN	0085	M	0003
	MATCH	0039	MSGSM	0003	MTP	0028	MTYPE	0020
	NBYTE	002E	OTADR	0006	PARMS	0031	PASHX	0006
R	PC	0007	PS	0043	PSTAT	0005	PUTBUF	0005
R	S	0004	SM	0068	SRSMEM	007C	START	0007
	TI	003A	WS	0039	X	0002		

0000 ERRORS IN JCDMPM

DIGITAL SYSTEMS LABORATORY
COMPUTER SCIENCE DEPARTMENT
CONCORDIA UNIVERSITY - MONTREAL QUEBEC

PROGRAM MNEMONIC: DMPMEM | REVISION LEVEL: R2.0/V** | REV DAT: 2

- DMPMEM - ENABLES DUMPING ALL MEMORY

DMPMEM - DUMP MEMORY BETWEEN LIMITS — PAGE

```
                0001          IDT   DMPMEM
                0002          DEF   DMPMEM
                0003  *
                0004  *REGISTER AND OTHER EQUATES
0000            0005  A       EQU   0
0001            0006  E       EQU   1
0002            0007  X       EQU   2
0003            0008  M       EQU   3
0004            0009  S       EQU   4
0005            0010  L       EQU   5
0006            0011  B       EQU   6
0007            0012  PC      EQU   7
0001            0013  BR      EQU   1
                0014  CALL    OPD   >0380,3
0000            0015  IOWMAN  EQU   0
```

```
                0018  REGS   BSS   7          CALLER'S REGISTERS
0007  0000      0019  DMPMEM DATA  0,0
0009  D8E0      0020         GSRF  REGS       SAVE CALLER'S REGS
0008  C530      0021         RMO   M,B        PTS TO PARMS BUILT BY JCDMPMEM
                0022 *PARMS: # WORDS TO DUMP, MEMORY LOC TO DUMP FROM,
                0023 *         STARTING ADR FOR LISTING, OFFSET FOR NEW LINE ADR
000C  1800      0024         @LDM  =PRB0
000E  C3B0      0025         CALL  IORMAN
000F  003F      0026         LDA   PRB0       GET STATUS FLAGS ON RETURN
0010  0811      0027         TABO  1          ANY ERRORS ON OPEN?
0011  7803      0028         BRU   DUMP       NO, DUMP MEMORY
0012  D8A0      0029  RETN   @LRF  REGS       RESTORE CALLER'S REGS
0014  7CF2      0030         BRU   *DMPMEM
```



```
0015  0032  DUMP   EQU   $
0015  0101  0033         LDA   1,BR       GET BUFFER ADR TO DUMP FROM
0016  8036  0034         STA   MEMLOC
0017  0103  0035         LDA   3,BR       GET INSTN FOR ADR OF DUMP LINE
0018  8020  0036         STA   OFS        SET INSTN TO INCR BY 8 (TI) OR 16 (SM)
              0037 *CALCULATE # LINES OF DUMP BY MAX(1,CEILING(# WORDS/8))
0019  0100  0038         LDA   0,BR       GET # WORDS TO DUMP
001A  CA03  0039         ARA   3
001B  CC00  0040         SZE   A          ENSURE THAT # LINES >=1
001C  CCE0  0041         SPL   A
001D  0701  0042         LDA   =1         USE A DEFAULT OF 1 LINE
001E  802F  0043         STA   NOLIN      # LINES OF DUMP TO BE PRINTED
001F  1902  0044         LDM   2,BR       GET STARTING ADR OF DUMP
0020  C534  0045 NEWLN   RMO   M,S        STARTING DUMP ADDR
0021  0800  0046         @LDE  =OUTBUF    OUTPUT BUFFER TO FORMAT DUMP
0023  C516  0047         RMO   E,B        B PTS TO WHERE TO STORE
0024  17F7  0048         LDX   =-9        LINE OF DUMP HAS ADDR+8 MEMORY LOCNS
              0049 *PRINTS ADDRESS, 8 CONSECUTIVE MEMORY LOCNS IN HEX, AND AS IS
0025  7805  0050         BRU   CON        GO TO CONVERT IT TO ASCII
0026  0426  0051  DMP2   LDA   *MEMLOC     GET CONTENTS OF ACTUAL MEMORY LOCN
0027  C503  0052         RMO   A,M        SAVE WORD CONTENTS
0028  5024  0053         IAO   MEMLOC     PT IT TO NEXT MEMORY LOCATION
              0054 *PRINT EACH MEMORY LOCATION AS IS, IF NOT PRINTABLE SUBSTITUTE
0029  7017  0055         BRL   STPRNT
002A  7016  0056         BRL   STPRNT
002B  8250  0057         STA   PRCHRS+8,X STORE WORD READY TO PRINT
002C  7850  0058  CON    BRU   BINHEX     CONVERTS BINARY TO HEX ASCII
002D  A100  0059         DST   0,BR       STORE 2 WORDS IN OUTPUT BUFFER
002E  0703  0060         LDA   =3         WIDTH OF FIELD, 2 WORDS+SPACE
002F  C3B6  0061         RAD   A,B        PT B TO NEXT FREE POSN IN OUTBUF
0030  40F5  0062         BIX   DMP2       IS ONE LINE COMPLETE?
0031  1800  0063         @LDM  =PRBL      YES, WRITE LINE PRB
0033  C380  0064         CALL  IORMAN     PRINT A LINE
0034  0810  0065         LDA   PRBL       GET STATUS FLAGS FROM PRB(1)
0035  B801  0066         TABZ  1          ERROR BIT SET?
0036  780C  0067         BRU   XIT        YES
0037  B803  0068         TABZ  3          IGNORE BIT SET?
0038  7804  0069         BRU   XIT        YES
0039        0070  OFS    BSS   1          LOCN PLUGGED WITH LDM =8 OR 16 INSTN
```

DMPMEM - DUMP MEMORY BETWEEN LIMITS PAGE 0004

```
003A  C0C3  0071          RAD   S,M       ADDR OF DUMP FOR NEW LINE
003B  4812  0072          DWT   NOLIN     ANY MORE LINES OF DUMP?
003C  78E3  0073          BRU   NEWLN     YES
003D  1900  0074  XIT     @LDM  =PRBC     SET TO CLOSE PRB
003F  C380  0075          CALL  IORHAN
0040  78D1  0076          BRU   RETN
            0077  *
0041  CA08  0078  STPRNT  CRA   8         FLIP BYTES
0042  3780  0079          IOR   =>80      SET TOP BIT OF BYTE FOR COMPARE
0043  67A0  0080          CPL   =>A0      CHECK LOWER LIMIT OF PRINTABLE CHARS
0044  C0C0  0081          SLE             IS IT PRINTABLE?
0045  7803  0082          BRU   PLUG      NO, PLUG IT WITH A DEFAULT
0046  67DF  0083          CPL   =>DF      MAYBE, CHECK UPPER LIMIT
0047  CD00  0084          SLT             IS CHAR PRINTABLE?
0048  C557  0085          RWO   L,PC      YES
0049  3800  0086  PLUG    @AND  =>FF00    MASK OUT BITS 8-15
004A  37AE  0087          IOR   =>AE      PLUG A '.'
004C  C557  0088          RWO   L,PC
P 004D      0089  MEMLOC  BSS   1
P 004E      0090  NOLIN   BSS   1
004F  0020  0091  PRBD    DATA  >20,7,75
0052  0020  0092  PRBL    DATA  >20,2,74,OUTBUF
0056  0020  0093  PRBC    DATA  >20,9
0058  8080  0094  OUTBUF  DATA  >8000,0000,0000,0000,0000,0000,0000,0000,0000
0072  A0A0  0095          DATA  '  '
P 0074      0096  PRCHRS  BSS   8
007C  8DBA  0097          DATA  >8DBA
```


BINHEX - CONVERTS BINARY TO HEX ASCII PAGE 0005

```
            0100          DEF   BINHEX
            0101  *SUBROUTINE BINHEX CONVERTS A BINARY WORD TO A STRING OF 4
            0102  *ASCII CODED HEXADECIMAL DIGITS
            0103  *ENTRY - M REG CONTAINS THE WORD TO BE CONVERTED
            0104  *EXIT - 4 ASCII CHARACTERS ARE IN A + E REGS
            0105  *
007D        0106  BINHEX  EQU   $
007D  9014  0107          STX   SAVX      SAVE CALLER'S X REG
007E  C542  0108          RWO   S,X       GET S REG
007F  9011  0109          STX   SAVS      SAVE IT
0080  B012  0110          DLD   ASCZ      SET A + E REGS TO ASCII ZEROS
0081  17FC  0111          LDX   =-4       4 TIMES TO LOOP, 4 CHARS/WORD
0082  C534  0112  MERGLP   RWO   M,S       COPY BINARY TO S REG
0083  C784  0113          REX   A,S       SWAP TO USE A FOR AND
0084  3F0F  0114          AND   =>F       ISOLATE ONE CHAR IN A
0085  670A  0115          CPL   =10       GUARANTEED <16, CHECK LOWER BOUND
0086  CD40  0116          SGT             IS IT BETWEEN A + F?
0087  2707  0117          ADD   =7        YES, ADD 7 TO GET ASCII A-F
0088  C784  0118          REX   A,S       ANYWAY SWAP BACK INTO S
0089  C0C1  0119          RAD   S,E       PUT ASCII CHAR INTO RIGHT POSN
008A  C8C8  0120          CRD   8         SHIFT A + E RIGHT 1 BYTE
008B  CA04  0121          CRM   4         MOVE 4 BITS DOWN TO 12-15
008C  40F5  0122          BIX   MERGLP    HAVE WE CONVERTED 4 CHARS?
008D  1003  0123          LDX   SAVS      GET CALLER'S S REG
008E  C524  0124          RWO   X,S       RESTORE IT
008F  1002  0125          LDX   SAVX      RESTORE X REG
0090  C557  0126          RWO   L,PC      RETURN TO CALLER
P 0091      0127  SAVS    BSS   1         SAVE CALLER'S S REG
P 0092      0128  SAVX    BSS   1         SAVE CALLER'S X REG
0093  8080  0129  ASCZ    DATA  '0000'
0000        0130          END
```

DIGITAL SYSTEMS LABORATORY
COMPUTER SCIENCE DEPARTMENT
CONCORDIA UNIVERSITY - MONTREAL QUEBEC

PROGRAM MNEMONIC	REVISION LEVEL	REVISION DATE
DMPMEM	R2.0/V**	2/12/

- DMPMEM - ENABLES DUMPING ALL MEMORY

BINHEX - CONVERTS BINARY TO HEX ASCII PAGE 0006

A	0000	ASCZ	0093	B	0006	BINHEX	007D
BR	0001	CALL	0380	CON	002C	DMP2	0026
DMPMEM	0007	DUMP	0015	E	0001	ERRMAX	0000
L	0005	M	0003	MEMLOC	004D	MEMGLP	0082
NEWLN	0020	NOLIN	004E	OFS	0039	OUTBUF	0058
PC	0007	PLUS	0049	PRBC	0056	PRBL	0052
PRBO	004F	PRCHRS	0074	REGS	0000	RETN	0012
S	0004	SAVS	0091	SAVX	00V2	STPRNT	0041
X	0002	XIT	0030				

0000 ERRORS IN DMPMEM

T.I. 980 SYMBOLIC ASSEMBLER RELEASE 2.0 VERSION *F

PAGE 0001

```
                    0001          IDT   LODMEM
                    0002  *
0000                0003  A       EQU   0
0001                0004  E       EQU   1
0002                0005  X       EQU   2
0003                0006  M       EQU   3
0004                0007  S       EQU   4
0005                0008  L       EQU   5
0006                0009  B       EQU   6
0007                0010  PC      EQU   7
0008                0011  ST      EQU   8
0001                0012  BR      EQU   1
                    0013  *
                    0014  SVC     OPD   >C380,3
                    0015  CALL    OPD   >C380,3
0007                0016  OTADR   EQU   7
0008                0017  OTINST  EQU   8
0000                0018  NOP     EQU   0          MOVE  AUX,AUX    INSTN IN SMS
                    0019  *
                    0020  *INPUTS OBJECT RECORDS FROM LUN >1A, OUTPUTS LISTING TO #20
0000 001A           0021  PRBIOP  DATA  >1A,8,72   OPEN REWIND ON INPUT
0003 0020           0022  PRBOOP  DATA  >20,8,75   OPEN REWIND ON OUTPUT
0006 001A           0023  PRBRD   DATA  >1A,0,0,BUFLIN  READ ASCII
0009 0020           0024  PRBWR   DATA  >20,2,0,BUFLIN  WRITE ASCII
000E 001A           0025  PRBICL  DATA  >1A,V      CLOSE ON INPUT
0010 0020           0026  PRBOCL  DATA  >20,10     CLOSE EOF ON OUTPUT
                    0027  *
0012 D400           0028  TAPEC   DATA  >D400      T CHAR, END OF OBJECT
0013 A000           0029  BLNK    DATA  >A000      BLANK CHAR, LEFT JUST.
0014 8D8A           0030  CRLF    DATA  >8D8A
0015 FF00           0031  LFBMSK  DATA  >FF00      LEFTMOST BYTE MASK
0016 2000           0032  SMADR   DATA  >2000
P 0017              0033  PSADR   BSS   1          LOCAL VAR TO KEEP TRACK OF INTFCE ADR
P 0018              0034  POBJ    BSS   1          LOCAL PTR TO OBJECT SAVE AREA
P 001V              0035  SADR    BSS   1          STARTING ADR AT WHICH PGM WAS LOADED
P 001A 30FA         0036  JTERM   DATA  TERM       TERMINATE SECTION
P 001B              0037  BUFLIN  BSS   38
                    0038  *
0041                0039  LODMEM  EQU   $
```

```
0041 1800 0040          @LDM  =PRBIOP
     0000 0041          BRS   PRBIOP
0043 C536 0042          RMO   M,B          SET FOR BASE REGISTER ADR
0044 C380 0043          SVC   0            OPEN REWIND ON INPUT
0045 1800 0044          @LDM  =PRBOOP
0047 C380 0045          SVC   0            OPEN REWIND ON OUTPUT
0048 07FF 0046          LDA   =-1          FLAG TO SIGNIF ADR NOT OUTPUT
004A 804D 0047          STA   PSADR        1ST TIME FORCE OUTPUT OF ADR
004A 0000 0048          @LDA  =OBJCOD      PTR TO SAVE AREA
004C 804B 004A          STA   POBJ         SET PTR TO ITS START
                0050 *
004D 1800 0051 READA    @LDM  =PRBRD
004F C380 0052          SVC   0            READ ASCII FROM INPUT
0050 0085 0053          LDA   PRBRD        GET STATUS FLAGS
0051 0B01 0054          TAB2  1            ERROR ON READ?
0052 78C7 0055          BRU   JTERM        YES
0053 0B02 0056          TAB2  2            EOF ENCOUNTERED?
0054 79FA 0057          BRU   TERM
0055 00B2 0058          LDA   PRBRD+2      GET # CHARS READ
0056 8085 0059          STA   PRBWR+2      WRITE SAME #
0057 0B33 0060          TABO  3,PRBOOP     WAS DEVICE REWINDABLE?
0059 7813 0061          BRU   STAREC       YES
                0062 *OTHERWISE APPEND CRLF
005A 00AD 0063          LDA   PRBRD+2      GET # CHARS READ
005B C501 0064          RMO   A,E          SAVE FOR EVEN-ODD
005C 2702 0065          ADD   =2           2 MORE FOR CRLF
005D 80AE 0066          STA   PRBWR+2      PLUG OUTPUT PRB
005E C801 0067          ARA   1            # WORDS READ+1
005F C502 0068          RMO   A,X          OFFSET INTO BUFFER, PTS TO END
0060 CC41 0069          SOD   E            IS IT ON WORD BOUNDARY?
0061 7804 0070          BRU   WDBDY
0062 32B7 0071          LDA   BUFLIN-1,X   GET LAST WORD IN BUFFER
0063 38B1 0072          AND   LFBMSK       TAKE ONLY LEFT BYTE
0064 3000 0073          @IOR  =>8D         PUT IN A CR
0066 32B3 0074          STA   BUFLIN-1,X   STORE IT BACK
0067 0000 0075          @LDA  =>8A00       GET LEFT LF
0069 82B1 0076          STA   BUFLIN,X     APPEND TO END
006A 7802 0077          BRU   STAREC
006B 00A6 0078 WDBDY    LDA   CRLF         GET 1 WORD
```



```
006C 82AD 0079          STA   BUFLIN-1,X   APPEND TO END OF BUFFER
                0080 *
006D 00A0 0081 STAREC   LDA   BUFLIN       GET 1ST WORD
006E 38A6 0082          AND   LFBMSK       EXAMINE LEFT BYTE ALONE
006F 60A3 0083          CPL   BLNK         IS IT A COMMENT?
0070 CDA0 0084          SNE                NO
0071 79F3 0085          BRU   ECHOP        PRINT COMMENT
0072 60AF 0086          CPL   TAPEC        IS IT END OF OBJECT?
0073 CDA0 0087          SNE                NO, ASSUME IT'S AN ADR
0074 7B7E 0088          BRU   ECHOP        JUST PRINT IT
                0089 *
                0090 *CONVERT ADDRESS, 5 OCTAL DIGITS
0075 00A5 0091          LDA   BUFLIN       GET 1ST WORD AGAIN
0076 C868 0092          LRD   8            SEPARATE BYTES IN A & E
0077 3F07 0093          AND   =7           CONVERTS ASCII TO BINARY
0078 C8C3 0094          LLA   3            SHIFT IN PREPN FOR NEXT DIGIT
0079 C503 0095          RMO   A,M          ACCUMULATE ADR IN M
007A C8E8 0096          LLD   8            GET 2ND DIGIT
007B 3F07 0097          AND   =7           ASCII TO BINARY
007C C4B0 0098          ROR   M,A          COMBINE WITH PREV DIGIT
007D C8C3 0099          LLA   3            SHIFT LEFT IN PREPN
007E C503 0100          RMO   A,M          HOLD IN M
007F 00AC 0101          LDA   BUFLIN+1     GET 2ND WORD OF ADR
0080 C868 0102          LRD   8            SEPARATE DIGITS
0081 3F07 0103          AND   =7           FORCE OCTAL BINARY
0082 C4B0 0104          ROR   M,A          COMBINE
0083 C8C3 0105          LLA   3            PREP FOR NEXT DIGIT
0084 C503 0106          RMO   A,M
0085 C8E8 0107          LLD   8            GET NEXT DIGIT
0086 3F07 0108          AND   =7           BINARY
0087 C4B0 0109          ROR   M,A
0088 C8C3 0110          LLA   3
0089 C503 0111          RMO   A,M
008A 00A2 0112          LDA   BUFLIN+2     LAST DIGIT
008B C848 0113          LRA   8            ISOLATE DIGIT
008C 3F07 0114          AND   =7
008D C4B0 0115          ROR   M,A          COMBINE ALL DIGITS IN A
                0116 *
                0117 *CHECK ADR FROM OBJECT AGAINST PSADR
```

```
0098  6800  0118          @CPA  =>3FF     CHECK THAT ADR IS IN RANGE
009D  CD80  0119          SGE             IS IT?
009E  780F  0120          BRU   FADR2       NO, BAD ADR
009F  1084  0121          LDX   PSADR       GET ADR AS ON INTERFACE
00A0  CCA2  0122          SNO   X           -1 IS FLAG, ADR NOT OUTPUT
00A1  7805  0123          BRU   FADR        1ST TIME FORCE ADR OUT
00A2  C002  0124          RSU   A,X         COMPARE ADRS
00A3  CC82  0125          SNZ   X           ARE THEY EQUAL?
00A4  7805  0126          BRU   CNVI        YES
00A5  0700  0127  OUTIN   LDA   =NOP        NO, FILL WITH NOPS
00A7  B518  0128          STA   *POBJ       STORE INSTN TO BE OUTPUT
00A9  5118  0129          INC   POBJ        PT TO NEXT
00AB  C389  0130          CALL  OTINST
00AC  5117  0131          INC   PSADR       UPDATE PTR TO MATCH INTFCE
00AD  40FA  0132          BIX   OUTIN       LOOP TILL DONE
00AF  7804  0133          BRU   CNVI
            0134  *
00B1  3117  0135  FADR    STA   PSADR       MAKE IT REFLECT INTERFACE ADR
00B2  B117  0136          STA   SADR        STARTING ADR OF OBJECT CODE
00B3  2116  0137  FADR2   ADD   SMADR      STARTING ADR FOR INTERFACE
00B4  C387  0138          CALL  OTADR       OUTPUT PS ADR
            0139  *
00B5  1703  0140  CNVI    LDX   =3          X PTS TO 1ST INSTN IN BUFFER
            0141  *
            0142  *CONVERT INSTRUCTION
00B6  0318  0143  CNVINS  LDA   BUFLIN,X    GET OPCODE FROM LINE BUFFER
00B7  C322  0144          RIN   X,X         NEXT WORD OF INSTN
00B8  3F07  0145          AND   =7          CONVERT ASCII # TO BINARY
00B9  6F07  0146          CPA   =7          IS IT A JMP?
00BA  CDA0  0147          SNE               NO -
00BB  7822  0148          BRU   JMPI        DECODE ADR AS 1 3 3 3 3, TYPE V
00BC  6F04  0149          CPA   =4          WHILE WE HAVE OPC, CHECK
00BD  C8C2  0150          LLA   2           SHIFT IN PREPN FOR NEXT DIGIT
00BE  C503  0151          RMO   A,M         ACCUMULATE IN 4
00BF  031B  0152          LDA   BUFLIN,X    GET NEXT WORD OF INSTN
00C0  C322  0153          RIN   X,X         NEXT WORD
00C1  3F03  0154          AND   =3          ONLY 2 BITS
00C2  C480  0155          ROR   M,A         COMBINE WITH OPC
00C3  C8C3  0156          LLA   3           PREPARE FOR NEXT OR
```



```
00C4  C503  0157          RMO   A,M         HOLD IN M
00C5  031B  0158          LDA   BUFLIN,X    GET NEXT WORD OF INSTN
00C6  C322  0159          RIN   X,X         NEXT WORD
00C7  C868  0160          LRD   8           GET LEFTMOST BYTE FIRST
00C8  3F07  0161          AND   =7          ASCII OCTAL DIGIT TO BINARY
00C9  C480  0162          ROR   M,A
00CA  CDC0  0163          SLE               IS OPC 4, 5, OR 6?
00CB  7802  0164          BRU   $+3         NO, THEN HAS TO BE TYPE I, II, IV
00CC  081B  0165          TABO  11          YES, IS SOURCE AN IV OR MS FIELD?
00CD  780B  0166          BRU   TYPIII      GET I FIELD 2 3 3
00CE  C8C3  0167          LLA   3           REG, SO GET ROTATE
00CF  C503  0168          RMO   A,M
00D0  C8E8  0169          LLD   8             RECOVER 2ND DIGIT
00D1  3F07  0170          AND   =7          GET 3 BIT ROTATE FIELD
00D2  C480  0171          ROR   M,A
00D3  C8C2  0172          LLA   2           DECODE AS 2 3
00D4  C503  0173          RMO   A,M
00D5  031B  0174          LDA   BUFLIN,X    GET NEXT 2 DIGITS
00D6  C868  0175          LRD   8           HANDLE LEFTMOST ONE
00D7  3F03  0176          AND   =3
00D8  781C  0177          BRU   LAST3
00D9  C8C2  0178  TYPIII  LLA   2
00DA  C503  0179          RMO   A,M
00DB  C8E8  0180          LLD   8
00DC  3F03  0181          AND   =3
00DD  7811  0182          BRU   L33
00DE  C8C1  0183  JMPI    LLA   1           DECODE ADR AS 1 3 3 3 3
00DF  C503  0184          RMO   A,M
00E0  031B  0185          LDA   BUFLIN,X
00E1  C322  0186          RIN   X,X
00E2  3F01  0187          AND   =1
00E3  C480  0188          ROR   M,A
00E4  C8C3  0189          LLA   3
00E5  C503  0190          RMO   A,M
00E6  031B  0191          LDA   BUFLIN,X
00E7  C322  0192          RIN   X,X
00E8  C868  0193          LRD   8
00E9  3F07  0194          AND   =7
00EA  C480  0195          ROR   M,A
```

```
00D9  C8C3  0196          LLA   3
00DA  C503  0197          RMO   A,M
00DB  C8E8  0198          LLD   8
00DC  3F07  0199          AND   =7
00DD  C480  0200  L33     ROR   M,A
00DE  C8C3  0201          LLA   3
00DF  C503  0202          RMO   A,M
00E0  D31B  0203          LDA   BUFLIN,X
00E1  C868  0204          LRD   8
00E2  3F07  0205          AND   =7
00E3  C322  0206  LAST3   XIN   X,X
00E4  C480  0207          ROR   M,A
00E5  C8C3  0208          LLA   3
00E6  C503  0209          RMO   A,M
00E7  C8E8  0210          LLD   8
00E8  3F07  0211          AND   =7
00E9  C480  0212          ROR   M,A       LEAVE WHOLE INSTN IN A
00EA  8518  0213          STA   *POBJ     STORE INSTN TO BE OUTPUT
00EB  5118  0214          IMO   POBJ      PT TO NEXT
            0215  *
            0216  *OUTPUT INSTRUCTION
00EC  C389  0217          CALL  OTINST    OUTPUT 1 INSTN
00ED  5117  0218          IMO   PSADR     ADR OF NEXT INSTN
            0219  *
00EE  D108  0220          LDA   PRBRD+2   GET # CHARS READ
00EF  C801  0221          AHA   1         # WORDS TO MATCH X
00F0  C420  0222          RCA   X,A       ANY MORE INSTN WORDS IN BUFLIN?
00F1  CD80  0223          SGE             NO
00F2  7881  0224          BRU   CNVINS    YES, LOOP BACK
            0225  *
00F3  1800  0226  ECHOP   @LDM  =PRBWR
00F5  C380  0227          SVC   0         WRITE ASCII, ECHO PRINT LINE
00F6  D10A  0228          LDA   PRBWR     GET STATUS FROM WRITE.
00F7  DB01  0229          TABZ  1         ERROR?
00F8  7801  0230          BRU   TERM      YES, TERMINATE
00F9  7A4D  0231          BRU   READA     LOOP UNTIL EOF
            0232  *
00FA  1800  0233  TERM    @LDM  =PRBCL
00FC  C3D0  0234          SVC   0         CLOSE INPUT
```



```
00FD  1800  0235          @LDM  =PRBOCL
00FF  C380  0236          SVC   0         CLOSE, WRITE EOF ON OUTPUT
0100  C381  0237          SVC   1         RETURN TO SYSTEM
            0238  *
0101        0239  OBJCOD  BSS   1024      MAXM IS 1K
      0041  0240          END   LODMEM
```

```
T.I. 990 SYMBOLIC ASSEMBLER          RELEASE 2.0  VERSION *F

                                                                PAGE  0008

A       0000    B       0006    BLNK    0013  R BR      0001
BUFLIN  0018    CALL    0380    CNVI    00A3    CNVINS  00A8
CRLF    0014    E       0001    ECHOP   00F3    FADR    00FF
FADR2   00A1    JMPI    00C3    JTERM   001A  R L       0005
L33     000C    LAST3   00E3    LFBMSK  0015    LODMEM  0041
M       0003    NOP     0000    OBJCOD  0101    OTADR   0007
OTINST  0008    OUTIN   0098  R PC      0007    POBJ    0018
PRBIOL  00DE    PRBIOP  0000    PRBUCL  0010    PRBOOP  0003
PRBRD   00D6    PRBWR   00DA    PSADR   0017    READA   004D
R S     0004    SADR    0019    SMADR   0016  R ST      0008
STAREC  005D    SVC     0380    TAPEC   0012    TERM    00FA
TYPIII  0007    WOBDY   0068    X       0002

0000 ERRORS IN LODMEM
```

```
DIGITAL SYSTEMS LABORATORY                          PROGRAM   REVISION   REVISION
COMPUTER SCIENCE DEPARTMENT                         MNEMONIC  LEVEL      DATE
CONCORDIA UNIVERSITY - MONTREAL QUEBEC              SMSUTL    R1.0/V**   27/02/78

          - SMSUTL - SMS SYSTEM UTILITY PACKAGE

     SMSUTL: OTBYTE,OTADR,OTINST,INBYTE,CHEQ,OUTP,INP          PAGE  0001

                0001        IDT   'SMSUTL'
                0002        REF   MSGSM,MSGSMR
                0003        REF   JCSTRT
                0004        REF   BINHEX
                0006  *
     0000       0007  A     EQU   0
     0001       0008  E     EQU   1
     0002       0009  X     EQU   2
     0003       0010  M     EQU   3
     0004       0011  S     EQU   4
     0005       0012  L     EQU   5
     0006       0013  B     EQU   6
     0007       0014  PC    EQU   7
     0008       0015  ST    EQU   8
     0001       0016  BR    EQU   1
                0017  *
     0058       0018  DMCON EQU   >58
     0059       0019  DMDAT EQU   >59
                0020  *
0000 FFFF       0021  FLAGS DATA  >FFFF
     000F       0022  FADR  EQU   15
```

```
                    0025          DEF   INBYTE
0001  3600  0026  READB  DATA  >0600        SET BITS 5 & 6, RESET 7
                  0027  *
            0002  0028  INBYTE  EQU   $
0002  D82F  0029        TABZ  BADR,FLAGS  ARE WE TRYING TO READ FROM A BAD ADR?
0004  7C20  0030        BRU   *JERADR       YES
0005  00F8  0031        LDA   READB         CONTROL BITS FOR READ DATA
0006  D879  0032        WDS   DMDAT         WRITE CONTROL TO INTERFACE
0007  0000  0033        DATA  A
0008  D858  0034        RDS   DMCON         READ STATUS
0009  0000  0035        DATA  A
000A  DB17  0036        TABO  7             IS DEVICE READY?
000B  78FC  0037        BRU   $-3           NO, LOOP BACK
000C  D859  0038        RDS   DMDAT         YES, READ 1 BYTE FROM MEMORY
000D  0000  0039        DATA  A
000E  3FFF  0040        AND   =>FF          MASK OUT CONTROL BITS
000F  8013  0041        STA   IBYTE         HOLD ONTO BYTE READ
0010  0400  0042        @LDA  REGS+A        GET ADR AS ON INTERFACE
0012  2701  0043        ADD   =1
0013  3C00  0044        @ADD  SMLEN
0015  8400  0045        @STA  REGS+A
0017  CC80  0046        SNZ   A             DID WE WRAP AROUND?
0018  7807  0047        BRU   *WRAI         YES
0019  3800  0048        @AND  =>3FF         CHECK FOR /PS ADRS
001B  CC00  0049        SZE   A             ARE WE POINTING TO BAD ADR?
001C  7805  0050        BRU   *WRAI+2       NO
001D  D57F  0051        SMBO  BADR,FLAGS
001F  7802  0052        BRJ   *WRAI+2
0020  D8C0  0053  *WRAI  @SSB  OTADR
                  0054  *EXIT WITH A-REG. CONTAINING BYTE RIGHT JUSTIFIED
      0022  0000  0055        LDA   IBYTE
P     0023        0056  IBYTE  BSS   1
      0024  C557  0057        RMO   L,PC
P     0025  0100  0058  JERADR DATA  ERADR
```



```
                  0061  *CHEQ - CHECK THE ACCURACY OF THE TRANSFER
                  0062  *      DO 2 READS UNTIL VALUE MATCHES
                  0063  *      CHECK VALUE AGAINST ONE WRITTEN
                  0064  *
                  0065  *
                  0066  *
            0026  0067  CHEQ  EQU   $
0026  C554  0068        RMO   L,S           SAVE RETURN PT
0027  17F6  0069        LDX   =-10          # TIMES TO RETRY
0028  0400  0070  TSTA  @LDA  REGS+A        GET ADR OF NEXT
002A  2F01  0071        SUB   =1
002B  D8C0  0072        @SSB  OTADR         RESET ADR REG
002D  70D4  0073        BRL   INBYTE        READ BACK BYTE JUST WRITTEN
002E  803C  0074        STA   SBYTE         HOLD ONTO IT FOR COMPARE
002F  0400  0075        @LDA  REGS+A        GET ADR AGAIN
0031  2F01  0076        SUB   =1
0032  D8C0  0077        @SSB  OTADR         RESET
0034  70CD  0078        BRL   INBYTE        READ A 2ND TIME
0035  6835  0079        CPA   SBYTE         COMPARE BYTES FROM 2 READS
0036  CDA0  0080        SNE                 ARE THEY THE SAME?
0037  7802  0081        BRJ   TECO          YES, CHECK THAT BYTE IS CORRECT
0039  40EF  0052        BIX   TSTA          NO, TRY READS AGAIN
0039  790E  0083        BRU   ERRD          ERROR ON 10TH READ
003A  C403  0084  TECO  RCA   A,M           COMPARE BYTE READ TO WRITTEN
003B  CDA0  0085        SNE                 ARE THEY THE SAME?
003C  C547  0086        RMO   S,PC
003D  CC82  0087        SNZ   X             RAN OUT OF ERRORS?
003E  7820  0088        BRU   ERRR          FLAG IT AS ERROR ON WRITE
003F  0400  0089        @LDA  REGS+A        GET NEXT ADR
0041  2F01  0090        SUB   =1            PT IT BACK
0042  D8C0  0091        @SSB  OTADR         RESET ADR REG
0044  C530  0092        RMO   M,A           RESTORE BYTE OF INSTN
0045  704E  0093        BRL   OTBYTE        WRITE IT AGAIN
0046  C322  0094        RIN   X,X           COUNT # ERRORS
0047  7BE0  0095        BRU   TSTA          GO THROUGH WHOLE CHECK AGAIN
0048  1000  0096  ERRD  @LDX  =*RDY
004A  9022  0097        STX   ERMS3+1       PLUG ERROR MSG
004B  C503  0098        RMO   A,M           2ND READ
004C  7400  0099        @BRL  BINHEX
```

```
004E A026 0100          DST   RD2          PLUG ERROR MSG
004F 1818 0101          LDM   SBYTE        GET 1ST VALUE READ
0050 7400 0102          @BRL  BINHEX
0052 A01F 0103          DST   RD1
0053 1C00 0104 ERRC     @LDM  REGS+A       NEXT SMS ADR
0055 C733 0105          RDE   M,M          ADR OF ERROR
0056 7400 0106          @BRL  BINHEX
0058 A016 0107          DST   HADR
0059 1000 0108          @LDX  =ERMS3
005B D8C0 0109          @SSB  MSGSM
005D 0000 0110          DATA  0
005E C547 0111          RMO   S,PC
          0112 *
005F 1000 0113 ERWR     @LDX  ='WR'
0061 9008 0114          STX   ERMS3+1      PLUG ERROR MSG
0062 C503 0115          RMO   A,M          BYTE AS READ FROM SMS
0063 7400 0116          @BRL  BINHEX
0065 A00C 0117          DST   RD1
0066 1829 0118          LDM   CINST        INSTN TRIED TO BE OUTPUT
0067 7400 0119          @BRL  BINHEX
0069 A00B 0120          DST   RD2
006A 78E8 0121          BRU   ERRC
          0122 *
006B      0123 SBYTE    BSS   1
006C 000A 0124 ERMS3    DATA  10,'XX 0000: 0000  0000'
     0075 0125 RD2      EQU   ERMS3+9
     0072 0126 RD1      EQU   ERMS3+6
     000F 0127 HADR     EQU   ERMS3+3
```



```
          0130          DEF   OTINST
          0131 *
          0132 *OTINST - ACCEPTS A 16 BIT INSTRUCTION IN BINARY IN A REG
          0133 *         AND LOADS MEMORY LOCATION PSADR
          0134 *
          0135 *    SMADR - STARTING MEMORY ADDRESS
          0136 *    BYTAD2 - TOGGLE FOR LOADING MORE THAN 1 PAGE, I.E. 16-BIT
          0137 *             MEMORIES AS IN SMS
          0138 *             =0 FOR 8-BIT MEMORIES, =>2000 FOR 16-BIT
          0139 *
0077 2000 0140 SMADR  DATA  >2000        SMS HIGH SPEED MEMORY (PGM STORAGE) ADR
0078 2000 0141 BYTAD2 DATA  >2000        2ND BYTE ADDRESS OFFSET
          0142 *
     0079 0143 OTINST EQU   $
0079 0000 0144          DATA  0,0
007B 8014 0145          STA   CINST        COPY INSTN TO BE OUTPUT
007C C868 0146          LHD   8            SEPARATE INSTN INTO 2 BYTES
007D C503 0147          RMO   A,M          HOLD MS HALF FOR CHECKING
007E 7015 0148 WRIT     BRL   OTBYTE       OUTPUT LEFTMOST BYTE
007F 70A6 0149          BRL   CHEQ         CHECK THE TRANSFER                A,A  OMR
0080 0068 0150          LDA   PSADR        GET ADR JUST WRITTEN TO
0081 20F8 0151          ADD   BYTAD2       OFFSET TO 2ND PAGE
0082 2F01 0152          SUB   =1           SAME LOCN, 2ND HALF
0083 D8C0 0153          @SSB  OTADR        OUTPUT ADR OF 2ND BYTE
0085 0700 0154          LDA   =0           CLEAR A FOR SHIFT BACK
0086 C8E8 0155          LLD   8            GET 2ND BYTE, RIGHTMOST
0087 C503 0156          RMO   A,M          HOLD LS HALF IN M
0089 7008 0157          BRL   OTBYTE       OUTPUT IT
008A 709C 0158          BRL   CHEQ         CHECK THE TRANSFER                A,A  OMR
008C 00DE 0159          LDA   PSADR        GET ADR JUST WRITTEN TO ON PAGE 2
008D 28EC 0160          SUB   BYTAD2       BACK TO PAGE 1
          0161 *NEXT ADR OF PROGRAM STORAGE IS IN A-REG, READY FOR OUTPUT
008C D8C0 0162          @SSB  OTADR        RESET FOR LEFT BYTE
          0163 *
008E D880 0164          @LSB  OTINST
008F      0165 CINST    BSS   1
```

```
                    0165          DEF   OTBYTE
0091  0000  0169  RETN2  DATA  0
0092  0700  0170  WRITE  DATA  >0700        SET BITS 5 & 6 & 7
0093  0000  0171  DBYTE  DATA  0
                    0172  *
                    0094  0173  OTBYTE EQU   $
0094  D82F  0174          TMBZ  @ADR,FLAGS    ARE WE GOING TO WRITE TO A BAD ADR?
0096  7806  0175  ERAD2   BRU   ERADR        YES
0097  30FA  0176          IOR   WRITE         SET CONTROL BITS
0098  C785  0177          REX   A,L
0099  30F7  0178          STA   RETN2
009A  C550  0179          RMO   L,A           HAVING SAVED L, GET BACK A
009B  7010  0180          BRL   OTBYTA        OUTPUT 1 BYTE
009C  004C  0181          LDA   REGS+A        GET MEMORY ADR AS ON INTERFACE
009D  2701  0182          ADD   =1            PT IT TO NEXT LOCN, UPDATE
009E  3846  0183          AND   SMLEN
009F  3049  0184          STA   REGS+A
00A0  CC00  0185          SZE   A             SM & WRAPAROUND?
00A1  7803  0186          BRJ   CHKA          NO, CHECK FOR VALID /PS ADRS
00A2  D8C0  0187          @SSB  OTADR         YES, THEN UPDATE HARDWARE REG
00A4  7CEC  0188          BRU   *RETN2
00A5  3800  0189  CHKA    @AND  =>3FF         CHECK FOR /PS ADRS
00A7  CC00  0190          SZE   A             DID WE WRAP AROUND?
00A8  7CE8  0191          BRU   *RETN2
00A9  3B7F  0192          S@BO  BADR,FLAGS    YES, SET BAD ADR FLAG
00AB  7CE5  0193          BRU   *RETN2
                    0194  *OTBYTA- ENTRY POINT FOR OTADR, CONTROL ALREADY SET
                    00AC  0195  OTBYTA EQU   $
00AC  8032  0196          STA   CBYTE         SAVE IN CASE OF RETRY
00AD  D87A  0197  RTRY    MDS   DMDAI
00AE  0080  0198          DATA  >80
00AF  78FD  0199          BRU   $-2
00B0  D858  0200          RDS   DMCON
00B1  0000  0201          DATA  A
00B2  DB15  0202          TABO  5             ACK FROM DEVICE?
00B3  C557  0203          RMO   L,PC
                    0204  *
00B4  902B  0205          STX   SAVX
00B5  C532  0206          RMO   M,X
```



```
00B6  902A  0207          STX   SAVM
00B7  1000  0208          JLDX  =ERMS2        NO, ERROR
00B8  D8C0  0209          @SSB  MSGSMR
00B9  00DB  0210          DATA  MODNM2
                    0211  *CHAR KEYED IN IS RIGHT JUSTIFIED IN M
00BC  1021  0212          LDX   YES1          YES RESPONSE FOR RETRY
00BD  C623  0213          RCL   X,M           COMPARE TO KEYIN
00BE  CD20  0214          SEQ                 ARE THEY THE SAME?
00BF  7C22  0215          BRU   *PSTRT
00C0  0000  0216          @LDA  =>8000        RESET INTERFACE
00C2  D878  0217          MDS   DMCON
00C3  0000  0218          DATA  A
00C4  001A  0219          LDA   CBYTE         RETREIVE BYTE TO BE OUTPUT
00C5  1018  0220          LDX   SAVM
00C6  C523  0221          RMO   X,M
00C7  1018  0222          LDX   SAVX
00C8  78E4  0223          BRU   RTRY
                    0224  *
00C9  3011  0225  ERMS2   DATA  17,'NO RESPONSE FROM INTERFACE, RETRY'
00DB  CFD4  0226  MODNM2  DATA  'OTBYTE'
00DE  00D9  0227  YES1    DATA  >00D9         ASCII NULL Y
00DF  0000  0228  CBYTE   DATA  0             CONTROL & BYTE TO OUTPUT
00E0  0000  0229  SAVX    DATA  0
00E1  0000  0230  SAVM    DATA  0
```

```
                    0233          DEF   OTADR
                0002 0234  PSTRT  DATA  JCSTRT
X 00E2          0000
                     0235  *
                     0236  *OTADR - ACCEPTS A 16 BIT ADDRESS IN A-REG
                     0237  *        OUTPUTS ADR TO MEMORY INTERFACE
                     0238  *
    00E3 0300    0239  MSADR  DATA  >0300      SET BITS 6 & 7, RESET 5
    00E4 0500    0240  LSADR  DATA  >0500      SET BITS 5 & 7,RESET 6
    00E5 0000    0241  SMLEN  DATA  0
    00E6 CFD4    0242  MODNM1  DATA  'OTADR'
                     0243  *
P   00E9         0244  REGS   BSS   7          FOR CALLER'S REG FILE
        00E9     0245  PSADR  EQU   REGS+A     CONTAINS ADR LAST OUTPUT
        00F0     0246  OTADR  EQU   $
    00F0 0000    0247        DATA  0,0
                     0248  *SETS PSADR EQU REGS+A TO ADR TO BE OUTPUT
    00F2 D8E0    0249        @SRF  REGS
                     0250  *ENSURE THAT MEMORY ADR IS IN RANGE
    00F4 3800    0251        @AND  =>FC00    EXAMINE TOP 6 BITS OF ADR
    00F6 CC80    0252        SNZ   A         IS IT AN ADR BETWEEN 0-3FF?
    00F7 7820    0253        BRU   AOK       YES
    00F8 6800    0254        @CPA  =>2000    IS IT 2000-23FF?
    00FA CDA0    0255        SNE
    00FB 781C    0256        BRU   AOK
    00FC 6800    0257        @CPA  =>4000    IS IT 4000-43FF?
    00FE CDA0    0258        SNE             ADDRESS IS OUT OF RANGE
    00FF 7818    0259        BRU   AOK
    0100 1DE8    0260  ERADR  LDM   REGS+A    RETREIVE ADR PASSED
    0101 7400    0261        @BRL  BINHEX    CONVERT ADR TO HEX ASCII
    0103 A00B    0262        DST   MADR      PLUG IT IN MESSAGE
    0104 1000    0263        @LDX  =ERMSI
    0106 D8C0    0264        @SSB  MSGSM
P   0108 00E6    0265        DATA  MODNM1
    0109 7CD6    0266        BRU   *PSTRT
    010A 0000    0267  ERMSI  DATA  13,'ADDRESS '
    010F B0B0    0268  MADR   DATA  '0000'     PLUG ADR HERE
    0111 A0CF    0269        DATA  ' OUT OF RANGE'
         0118    0270  AOK    EQU   $
```



```
    0113 3000    0271        @IOR  =>3FF
    011A B0CA    0272        STA   SMLEN
    011B 0800    0273        @LDE  =>8000    RESET INTERFACE
    011D D878    0274        MDS   DMCON
    011E 0001    0275        DATA  E
                     0276  *
    011F 00C9    0277        LDA   REGS+A    RETREIVE ADR TO BE OUTPUT
    0120 C868    0278        LRD   8             KEEP MSB OF MEMORY ADR
    0121 30C1    0279        IOR   MSADR         SET CONTROL BITS FOR INTERF
    0122 7089    0280        BRL   OTBYTA        OUTPUT TOP BYTE OF ADR
    0123 0700    0281        LDA   =0            PREPARE FOR SHIFT BACK
    0124 C8E8    0282        LLD   8             GET LSB OF MEMORY ADR
    0125 30E6    0283        IOR   LSADR         SET CONTROL BITS
    0126 7085    0284        BRL   OTBYTA        OUTPUT 2ND BYTE OF ADR
    0127 D86F    0285        SMBZ  BADR.FLAGS
    0129 D8A0    0286        @LRF  REGS          RESTORE CALLER'S REGS
    012B D880    0287        @LSB  OTADR
```

INP - INPUTS FROM SMS MEMORY PAGE 0010

```
                    0290          DEF   INP
                    0291  *
                    0292  *A GETS CHAR INPUT, E COUNTS, X +VE # CHARS, B PTS BUFFER, L SUBR
                    0293  *
0120  0800          0294  INP     DATA  0,0
0121  0F00          0295          LDE   =0        # CHARS READ, FLIP FLOP FOR PACK
0122  7400          0296  NBYTE   @BRL  INBYTE    INPUT 1 BYTE FROM SMS MEMORY
0123  C311          0297          RIN   E,E       CHAR COUNT
0124  CC41          0298          SOD   E         TOP HALF OF WORD?
0125  7803          0299          BRU     BYTE2           NO
0126  C808          0300          LLA     8               YES, SHIFT IT
0127  8100          0301          STA     0,BR            STORE IT AT CURRENT POSN
0128  7803          0302          BRU     CHKMO
0129  3100          0303  BYTE2   IOR     0,BR            COMBINE WITH PREVIOUS CHAR
012A  8100          0304          STA     0,BR            STORE IT BACK IN BUFFER
012B  C366          0305          RIN     B,B             FULL WORD, SO PT B TO NEXT
012C  40F4          0306  CHKMO   BIX     NBYTE   ANY MORE?
012D  D880          0307          @LSB  INP
```

DIGITAL SYSTEMS LABORATORY
COMPUTER SCIENCE DEPARTMENT
CONCORDIA UNIVERSITY - MONTREAL QUEBEC

PROGRAM MNEMONIC: SMSUTL | REVISION LEVEL: R1.0/V** | REVISION DATE: 27/02/78

- SMSUTL - SMS SYSTEM UTILITY PACKAGE

OUTP - OUTPUTS TO SMS MEMORY PAGE 0011

```
                    0310          DEF   OUTP
                    0311  *
012E  0707          0312  WRITB2  DATA  >0707     SET BITS 5 & 6 & 7 IN BOTH HALVES
                    0313  *
                    0314  *A GETS 1 WORD, E HOLDS 2ND BYTE, X -VE # CHARS, B PTS BUFFER, L SUBR
                    0315  *
012F  0000          0316  OUTP    DATA  0,0
0130  3100          0317  NWRD    LDA     0,BR            GET ONE WORD
0131  C366          0318          RIN     B,B             ALREADY PT TO NEXT
0132  38FA          0319          LDE   =WRITB2   SET CONTROL BITS FOR WRITE
0133  C9C8          0320          CRD   8         LEFT CHAR INA, NEXT IN E
0134  CA28          0321          CRE   8         MAKE IT CONTROL & THEN CHAR
0135  7400          0322  OTB     @BRL  OTBYTE
0136  4002          0323          BIX     NCHR            ANY MORE CHARS?
0137  D880          0324          @LSB  OUTP
0138  CC81          0325  NCHR    SNZ     E               2ND BYTE YET TO PROCESS?
0139  79F4          0326          BRU     NWRD            NO, GET NEW WORD
013A  C510          0327          RMO   E,A       GET 2ND CHAR OF WORD
013B  0F00          0328          LDE   =0        SET E TO FORCE NEW WORD
013C  78F6          0329          BRU     OTB
      0000          0331          END
```

DIGITAL SYSTEMS LABORATORY
COMPUTER SCIENCE DEPARTMENT
CONCORDIA UNIVERSITY - MONTREAL QUEBEC

PROGRAM MNEMONIC	REVISION LEVEL	REVISION DATE
SMSUTL	R1.0/V**	27/02/78

- SMSUTL - SMS SYSTEM UTILITY PACKAGE

SYMBOL TABLE

	A	0000		AOK	011B		B	0006
	BADR	000F		BINHEX	0003		BR	0001
	BYTE2	0138		BYTE	00AF		CHEQ	3026
	CHKA	00A5		CHKMD	012B		CINST	00V0
R	DBYTE	00V3		DMCON	00B9		DMDAT	0059
	E	0001	R	ERAD2	00V6		ERADR	0130
	ERMS1	010A		ERMS2	00C9		ERMS3	004C
	ERRC	0053		ERRD	0049		ERR	00BF
	FLAGS	0000		HADR	006F		IBYTE	0023
	INBYTE	0002		INP	012D		JCSTRT	0002
	JERADR	0025		L	0005		LSADR	00E4
	M	0003		MADR	010F		MODNM1	00E6
	MODNM2	00D8		MSADR	00E3		MSGSM	0000
	MSGSMR	0001		NBYTE	0130		NCHR	0148
	NWRD	0141		OTADR	00F0		OTB	0146
	OTBYTA	00AC		OTBYTE	00V4		OUTINST	007V
	OUTP	013F		PC	0007		PSADR	00E9
	PSTRT	00E2		RD1	0072		RD2	0075
	READB	0001		REGS	00EV		RETN2	00V1
	RTRY	00AD		S	0004		SAVM	0021
	SAVX	00B0		SBYTE	0068	R	SMADR	0077
	SMLEN	00E5	R	ST	0008		TECO	003A
	TSTA	0028		WRAI	0020		WRITB	0092
	WRITB2	013E	R	WRTA	007E		X	0002
	YES1	00DE						

0000 ERRORS IN SMSUTL

```
T.I. 980 SYMBOLIC ASSEMBLER          RELEASE 2.0  VERSION *F

                                                                    PAGE  0001

                    0001  *PROGRAM TO TEST COMMUNICATION, DOES OPEN, WRITE, CLOSE
                    0002        IDT   'TWRIT'
                    0003  SVC   OPD   >C380,3
          0000      0004  ST    EQU   $
0000 1800 0005              @LDM  =PRB0     DO OPEN ON COM/LOG
0002 C380 0006              SVC   0
0003 0009 0007              LDA   PRB0      GET STATUS FLAGS RETURNED
0004 0801 0008              TAEZ  1         CHECK FOR AN ERROR ON OPEN
0005 C381 0009              SVC   1         UNRECOVERABLE, END IT
0006 1800 0010              @LDM  =PRBW     WRITE TO COM/LOG
0008 C380 0011              SVC   0
0009 1800 0012              @LDM  =PRBC     DO CLOSE ON COM/LOG
000B C380 0013              SVC   0
000C C381 0014              SVC   1
                    0015  *
000D 0030 0016  PRB0  DATA  >30,3,80
0010 0030 0017  PRBW  DATA  >30,2,82,BUFOUT
0014 0030 0018  PRBC  DATA  >30,10
0016 D4C8 0019  BUFOUT DATA 'THIS IS AN OUTPUT MESSAGE PLACED IN SM, '
002A D3C8 0020        DATA  'SHOULD BE PRINTED ON THE CONSOLE BY SMS.'
003E 8D8A 0021        DATA  >8D8A
          0000 0022        END   ST
```

```
T.I. 980 SYMBOLIC ASSEMBLER          RELEASE 2.0  VERSION *F

                                                                    PAGE  0002

BUFOUT 0016      PRBC   0014      PRB0   000D      PRBW   0010
ST     0000      SVC    C380

0000 ERRORS IN TWRIT
```

```
T.I. 990 SYMBOLIC ASSEMBLER          RELEASE 2.0  VERSION *F

                                                          PAGE  0001

                     0001  *PROGRAM TO TEST COMMUNICATION, DOES OPEN, READ, CLOSE
                     0002        IDT   'TREAD'
                     0003  SVC   OPD   >C380,3
              0000   0004  ST    EQU   $
0000  1800          0005        @LDN  @PRBOR    DO OPEN,READ,CLOSE ON COM/LOG
0002  C380          0006        SVC   0
0003  0309          0007        LDA   PRBOR     GET STATUS FLAGS RETURNED
0004  0801          0008        TABZ  1         CHECK FOR AN ERROR ON OPEN
0005  C381          0009        SVC   1         UNRECOVERABLE, END IT
0006  1800          0010        @LDN  @PRBR     READ FROM COM/LOG
0008  C380          0011        SVC   0
0009  1800          0012        @LDN  @PRBC
000B  C380          0013        SVC   0
                     0014  *
000C  C381          0015        SVC   1
000D  0030          0016  PRBOR  DATA  >30,3,77
0010  0030          0017  PRBC   DATA  >30,9
0012  0030          0018  PRBR   DATA  >30,0,0,BUFIN
0016                 0019  BUFIN  BSS   40
              0000   0020        END   ST
```

```
T.I. 990 SYMBOLIC ASSEMBLER          RELEASE 2.0  VERSION *F

                                                          PAGE  0002

BUFIN  0016      PRBC   0010      PRBOR  000D      PRBR   0012
ST     0000      SVC    C380

0000 ERRORS IN  TREAD
```

```
                    0001          IDT   'TOPS4'
                    0002  *PROGRAM TO TEST SHARED MEMORY BUFFER FROM TI END- OPCODES
                    0003  X       EQU   2
                    0004  M       EQU   3
                    0005  B       EQU   6
                    0006  BR      EQU   1
                    0007          SVC   OPD   >C380,3
0000  17F4          0008  ST      LDX   #-12      # PRBS TO TEST
0001  1A0F          0009  IORO    LDA   PRBADR+12,X   GET ADR OF I PR3
0002  0380          0010          SVC   0
0003  40FD          0011          BIX   IORO      ANY MORE TO TEST?
0004  0381          0012          SVC   1
                    0013  *
                    0014  PRBADR  EQU   $
P 0005              0015          DATA  PRBRDO
P 0006              0016          DATA  PRBWRA
P 0007              0017          DATA  PRBWRO
P 0008              0018          DATA  PRBREA
P 0009              0019          DATA  PRBKSP
P 000A              0020          DATA  PRBFWD
P 000B              0021          DATA  PRBOPN
P 000C              0022          DATA  PRBOPR
P 000D              0023          DATA  PRBCLO
P 000E              0024          DATA  PRBCLF
P 000F              0025          DATA  PRBUNL
P 0010              0026          DATA  PRBRDA
                    0027  *
0011  0030          0028  PRBRDA  DATA  >30,0,0,BUFIN
0015  0030          0029  PRBRDO  DATA  >30,1,0,BUFIN
0019  0030          0030  PRBWRA  DATA  >30,2,76,BUFOUT
001D  0030          0031  PRBWRO  DATA  >30,3,80,BUFOUT
0021  0030          0032  PRBREA  DATA  >30,4
0023  0030          0033  PRBKSP  DATA  >30,5,25
0026  0030          0034  PRBFWD  DATA  >30,6,100
0029  0030          0035  PRBOPN  DATA  >30,7,85
002C  0030          0036  PRBOPR  DATA  >30,8,77
002F  0030          0037  PRBCLO  DATA  >30,9
0031  0030          0038  PRBCLF  DATA  >30,10
0033  0030          0039  PRBUNL  DATA  >30,11
```



```
                    0040  *
0035  C1C2          0041  BUFOUT  DATA  'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+-()'
0049  DEDF          0042          DATA  '^_"<[]@>?*!#$\%&/.,$=ABCDEFGHIJKLMNOPQR'
P 005D              0043  BUFIN   BSS   40
      0000          0044          END   ST
```

T.I. 9900 SYMBOLIC ASSEMBLER RELEASE 2.0 VERSION *F

R 0	0006	R BR	0001	BUFIN	005D	BUFOUT	0035
IORQ	0001	R M	0003	PRBADR	0005	PRBCLF	0031
PRBCLO	002F	PRBFMD	0025	PRBKSP	0023	PRBOPN	0029
PRBOPR	002C	PRBRDA	0011	PRBRDO	0015	PRBREW	0021
PRBUNL	0033	PRBWRA	0019	PRBWRO	001D	ST	0000
SVC	0330	X	0002				

0000 ERRORS IN TOPSM

```
 1                                  PROG  DREG
 2                          *DISPLAY REGISTER CONTENTS FOR R# SPECIFIED BY R:
 3                          *
 4          001 7 3         SW      LIV   1,7,3       BITS 5, 6, 7 OF IV BYTE
 5          001 7 0         DISPLAY LIV   1,7,8       IV BYTE 1
 6                          *
 7  00000   6 00000                 XMIT  0,AUX
 8  00001   6 01007                 XMIT  7H,R1
 9  00002   6 02042                 XMIT  42H,R2
10  00003   6 03035                 XMIT  35H,R3
11  00004   6 04017                 XMIT  17H,R4
12  00005   6 05340                 XMIT  340H,R5
13  00006   6 06000                 XMIT  0,R6
14  00007   6 11377                 XMIT  -1,R11
15                          *
16  00010   6 07001                 SEL   DISPLAY
17  00011   4 27313                 XEC   TABLE(DISPLAY,3),8
18  00012   7 00012                 JMP   *
19  00013   0 00027         TABLE   MOVE  AUX,DISPLAY
20  00014   0 01027                 MOVE  R1,DISPLAY
21  00015   0 02027                 MOVE  R2,DISPLAY
22  00016   0 03027                 MOVE  R3,DISPLAY
23  00017   0 04027                 MOVE  R4,DISPLAY
24  00020   0 05027                 MOVE  R5,DISPLAY
25  00021   0 06027                 MOVE  R6,DISPLAY
26  00022   0 11027                 MOVE  R11,DISPLAY
27                                  END   DREG
```

```
 1                                    PROG  DSREG
 2                              *DISPLAYS REG CONTENTS SPECIFIED BY 1 SWITCH, SW
 3                              *
 4                 001 7 0      REG     LIV   1,7,8          IV BYTE 1
 5                 001 0 1      BIT0    LIV   REG,0,1
 6                 001 1 1      BIT1    LIV   REG,1,1
 7                 001 2 1      BIT2    LIV   REG,2,1
 8                 001 3 1      BIT3    LIV   REG,3,1
 9                 001 4 1      BIT4    LIV   REG,4,1
10                 001 5 1      BIT5    LIV   REG,5,1
11                 001 6 1      BIT6    LIV   REG,6,1
12                 001 7 1      BIT7    LIV   REG,7,1
13                              *
14  00000   6 07001                     SEL    REG
15  00001   4 27103              TABS    XEC   T1(BIT7)
16  00002   7 00002                     JMP   *
17                              *
18  00003   4 26105              T1      XEC   T2(BIT6)
19  00004   0 11027                     MOVF  R11,REG
20  00005   4 25107              T2      XEC   T3(BIT5)
21  00006   0 06027                     MOVF  R6,REG
22  00007   4 24111              T3      XEC   T4(BIT4)
23  00010   0 05027                     MOVF  R5,REG
24  00011   4 23113              T4      XEC   T5(BIT3)
25  00012   0 04027                     MOVF  R4,REG
26  00013   4 22115              T5      XEC   T6(BIT2)
27  00014   0 03027                     MOVF  R3,REG
28  00015   4 21117              T6      XEC   T7(BIT1)
29  00016   0 02027                     MOVF  R2,REG
30  00017   4 20121              T7      XEC   T8(BIT0)
31  00020   0 01027                     MOVF  R1,REG
32  00021   7 00001              T8      JMP   TABS
33  00022   0 00027                     MOVF  AUX,REG
34                                      END   DSREG
```

```
                                    PROG  BUG
                                *FOR DEBUGGING.
                                *DISPLAYS REG OR WS CONTENTS REQUESTED FROM SWITCH
                                *IF BIT 0 IS NOT SET, THEN DISPLAY REGISTER CONTEN
                                *FOR R# SPECIFIED BY BITS 5, 6, 7 OF SWITCHES
                                *OTHERWISE ASSUME ADDRESS IS WORKING STORAGE AND D
                                *
          177 7 1               PSR     RIV   177H,7,1
          200 7 0               WS0     RIV   200H,7,8      1ST BYTE OF WORKING STOR
                                *
          001 7 0               DISPLAY LIV   1,7,8         IV BYTE 1
          001 7 0               SW      LIV   DISPLAY,7,8
          001 0 1               BIT0    LIV   SW,0,1        BIT 0 OF SWITCH
                                *
00000     6 07001                       SEL   SW
00001     5 20114                       NZT   BIT0,T1       IS IT A WS LOCN?
00002     4 27304                       XEC   TABLE(SW,3),8   NO, ASSUME REG
00003     7 00003                       JMP   *
00004     0 00027               TABLE   MOVF  AUX,DISPLAY
00005     0 01027                       MOVF  R1,DISPLAY
00006     0 02027                       MOVF  R2,DISPLAY
00007     0 03027                       MOVF  R3,DISPLAY
00010     0 04027                       MOVF  R4,DISPLAY
00011     0 05027                       MOVF  R5,DISPLAY
00012     0 06027                       MOVF  R6,DISPLAY
00013     0 11027                       MOVF  R11,DISPLAY   NOTE- SET 7 ON SWITCHES
                                *
                                *DISPLAY WORKING STORAGE
00014     6 17177               T1      SEL   PSR
00015     6 37100                       XMIT  0,PSR           SET REG TO PAGE 0
00016     0 27017                       MOVF  SW,IVR          GET ADR FROM SWITCHES
00017     0 37001                       MOVF  WS0,R1          TRANSFER BYTE OF WS TO
00020     0 01027                       MOVF  R1,DISPLAY      THEN TO LED READ-OUT
00021     7 00021                       JMP   *
                                        END   BUG
```

```
                                        PROG  RESET
                                *ZEROS ALL REGISTERS
                                *INITIALIZES WORKING STORAGE TO ZERO
                                *
          177 7 1               PSR     RIV   177H,7,1    BIT FOR PAGE SELECT
           000000               PAGE0   EQU   0
          200 7 0               WS0     RIV   200H,7,8    1ST BYTE OF WS, DUMMY
                                *
00000   6 17177                         SEL   PSR          PAGE SELECT
00001   6 37100                         XMIT  PAGE0,PSR
00002   6 03000                         XMIT  0,R3         INIT VALUE
00003   6 02200                         XMIT  WS0,R2       STARTING ADR
00004   6 01200                         XMIT  128,R1       NUMBER OF LOCATIONS
                                *
00005   0 02017               WRINC     MOVE  R2,IVR       SELECT CURRENT WS ADR
00006   0 03037                         MOVE  R3,WS0       ZERO THAT LOCATION
00007   6 00001                         XMIT  1,AUX        INCR
00010   1 02002                         ADD   R2,R2        PTS TO NEXT
00011   6 00377                         XMIT  -1,AUX       DECR
00012   1 01001                         ADD   R1,R1        # LOCATIONS LEFT
00013   5 01005                         NZT   R1,WRINC     LOOP UNTIL DONE
                                *
                                *R1  R3 ARE ALREADY ZERO
00014   6 00000                         XMIT  0,AUX
00015   6 02000                         XMIT  0,R2
00016   6 04000                         XMIT  0,R4
00017   6 05000                         XMIT  0,R5
00020   6 06000                         XMIT  0,R6
00021   6 17000                         XMIT  0,IVR
00022   6 11000                         XMIT  0,R11
00023   6 07000                         XMIT  0,IVL
00024   7 00024                         JMP   *
                                        END   RESET
```

```
INITWS      SMS MICROCONTROLLER ASSEMBLER   SMSASM/CDC VER 1.1      78/C

    1                                   PROG  INITWS
    2                           *INITIALIZES PAGE 0 OF WORKING STORAGE, EACH LOC
    3                           *
    4          177 7 1          PSR     RIV   177H,7,1
    5          200 7 0          WS0     RIV   128,7,8       1ST BYTE OF WORKING STO
    6          001 7 0          SW      LIV   1,7,8         IV BYTE 1
    7                           *
    8                           INC     MACRO D
    9                                   XMIT  1,AUX
   10                                   ADD   D,D
   11                                   ENDM
   12                           *
   13                           DEC     MACRO D
   14                                   XMIT  -1,AUX
   15                                   ADD   D,D
   16                                   ENDM
   17                           *
   18  00000   6 17177                  SEL   PSR
   19  00001   6 37100                  XMIT  0,PSR         SET REG TO PAGE 0
   20  00002   6 02200                  XMIT  WS0,R2        STARTING ADR OF WS
   21  00003   6 01200                  XMIT  128,R1        LENGTH OF 1 PAGE
   22  00004   0 02017          WRINC   MOVE  R2,IVR        SET ADR
   23  00005   0 02037                  MOVE  R2,WS0        INITIALIZE
   24                                   INC   R2            PT TO NEXT LOCN
   25                                   DEC   R1            1 LESS LOCN LEFT
   26  00012   5 01004                  NZT   R1,WRINC      ANY LEFT? YES, LOOP
   27  00013   7 00013                  JMP   *             NO
   28                                   END   INITWS
```

```
                                PROG  DWOKS
                    *DISPLAYS BYTE, CONTENT OF WORKING STORAGE SET O
                    *
          177 7 1   PSR   RIV   177H,7,1
          001 7 0   SW    LIV   1,7,8       IV BYTE 1
          200 7 0   WS0   RIV   128,7,8     1ST BYTE OF WORKING STOR
                    *
00000   6 17177           SEL   PSR
00001   6 37100           XMIT  0,PSR       SET REG TO PAGE 0
00002   6 07001           SEL   SW
00003   6 00200           XMIT  200H,AUX    ADR FOR WORKING STORAGE
00004   1 27017           ADD   SW,IVR      SELECT ADR REQUESTED BY
00005   0 37001           MOVF  WS0,R1      TRANSFER BYTE OF WS TO F
00006   0 01027           MOVF  R1,SW       DISPLAY CONTENTS
00007   7 00007           JMP   *
                          END   DWOKS
```

```
RDCONS          SMS MICROCONTROLLER ASSEMBLER   SMSASM/CDC VER 1.1      78/08

   1                                            PROG  RDCONS
   2                                 *PROGRAM TO READ 1 CHAR FROM ADM CONSOLE TO DISPL
   3                                 *
   4          001 7 0    DISPLAY     LIV   1,7,8      IV BYTE 1
   5          005 7 0    CONTROL     LIV   5,7,8      IV BYTE 5
   6          006 7 0    DATA        LIV   6,7,8      IV BYTE 6
   7          006 7 0    STATUS      LIV   6,7,8      SAME AS DATA
   8          006 5 3    ERRS        LIV   STATUS,5,3   PE, FE, OE ARE BITS
   9          006 7 1    DR          LIV   STATUS,7,1   DATA READY, BIT 7
  10                                 *
  11                                 NOP         MACRO
  12                                             MOVE    AUX,AUX    ALSO USES UP 300 NS
  13                                             ENDM
  14                                 *
  15  00000   6 07005                           SEL   CONTROL
  16  00001   6 00172                           XMIT  01111010B,AUX     MRESET,SFD,DATA T
  17  00002   0 00027                           MOVE  AUX,CONTROL       RESET CONSOLE INT
  18  00003   6 07006                           SEL   STATUS
  19  00004   0 27001                           MOVE  STATUS,R1
  20                                 *
  21  00005   6 07005    RD2         SEL   CONTROL
  22  00006   6 00063                           XMIT  00110011B,AUX     DATA IN,RRD
  23  00007   0 00027                           MOVE  AUX,CONTROL       READ STATUS
  24  00010   6 07006                           SEL   STATUS
  25  00011   0 27002                           MOVE  STATUS,R2
  26  00012   4 27112                           XEC   *(DR)     WAIT FOR CHAR TO BE KEYED
  27  00013   5 25324                           N7T   ERRS,ERHAN    CHECK FOR ERRORS
  28  00014   0 27003                           MOVE  STATUS,R3
  29  00015   6 07005                           SEL   CONTROL
  30  00016   6 00070                           XMIT  00111000B,AUX     SFD,DATA IN,DRR
  31  00017   0 00027                           MOVE  AUX,CONTROL       READ CHAR INTO DA
  32  00020   6 07006                           SEL   DATA              ACCESS TO CHAR
  33                                             NOP     ENSURE 500 NS BEFORE ACCESSING D
  34  00022   0 27004                           MOVE  DATA,R4           DISPLAY IT
  35  00023   7 00030                           JMP   DREG
  36  00024   0 27000    ERHAN       MOVE  STATUS,AUX
  37  00025   6 07001                           SEL   DISPLAY
  38  00026   0 00027                           MOVE  AUX,DISPLAY
  39  00027   7 00027                           JMP   *
  40                                 *DISPLAYS REG CONTENTS SPECIFIED BY 1 SWITCH, SWO
  41                                 *
  42          001 0 1    BIT0        LIV   DISPLAY,0,1
  43          001 1 1    BIT1        LIV   DISPLAY,1,1
  44          001 2 1    BIT2        LIV   DISPLAY,2,1
  45          001 3 1    BIT3        LIV   DISPLAY,3,1
  46          001 4 1    BIT4        LIV   DISPLAY,4,1
  47          001 5 1    BIT5        LIV   DISPLAY,5,1
  48          001 6 1    BIT6        LIV   DISPLAY,6,1
  49          001 7 1    BIT7        LIV   DISPLAY,7,1
  50                                 *
  51  00030   6 07001    DREG    SEL   DISPLAY
  52  00031   7 00040            ORG   32,32
  53  00040   4 27102    TABS    XEC   T1(BIT7)
  54  00041   7 00005            JMP   RD2
  55                                 *
  56  00042   4 26104    T1      XEC   T2(BIT6)
  57  00043   0 11027            MOVF  R11,DISPLAY
  58  00044   4 25106    T2      XEC   T3(BIT5)
```

```
 59  00045  0 06027          MOVE  R6,DISPLAY
 60  00046  4 24110  T3      XEC   T4(BIT4)
 61  00047  0 05027          MOVE  R5,DISPLAY
 62  00050  4 23112  T4      XEC   T5(BIT3)
 63  00051  0 04027          MOVE  R4,DISPLAY
 64  00052  4 22114  T5      XEC   T6(BIT2)
 65  00053  0 03027          MOVE  R3,DISPLAY
 66  00054  4 21116  T6      XEC   T7(BIT1)
 67  00055  0 02027          MOVE  R2,DISPLAY
 68  00056  4 20120  T7      XEC   T8(BIT0)
 69  00057  0 01027          MOVE  R1,DISPLAY
 70  00060  7 00040  T8      JMP   TABS
 71  00061  0 00027          MOVE  AUX,DISPLAY
 72                            END   RDCONS
```

```
T2CON          SMS MICROCONTROLLER ASSEMBLER   SMSASM/COC VER 1.1      78/08

                                        PROG  T2CON
                                *PROGRAM TO OUTPUT 2 CHARS CONSEQUITIVELY TO CONSO
                                *
                 001 7 0    DISPLAY  LIV   1,7,8      IV BYTE 1
                 002 7 0    CONTROL  LIV   2,7,8      IV BYTE 2
                 002 3 1    TBRL     LIV   CONTROL,3,1
                 003 7 0    DATA     LIV   3,7,8      IV BYTE 3
                 003 7 0    STATUS   LIV   3,7,8      SHARES IV BYTE WITH DATA
                 003 5 3    ERRS     LIV   STATUS,5,3    PE, FE, OE ARE BITS
                 003 6 1    TBRE     LIV   STATUS,6,1    XMIT BUFFER EMPTY, R
                                *
                 200 7 0    CHAR1    RIV   200H,7,0
                 201 7 0    CHAR2    RIV   201H,7,0
                 202 7 0    CR       RIV   202H,7,0
                 203 7 0    LF       RIV   203H,7,0
                                *
                            NOP      MACRO
                                     MOVE    AUX,AUX    ALSO USES UP 300 NS
                                     ENDM
                                *
00000  6 07002                       SFL   CONTROL
00001  6 00136                       XMIT  01011110B,AUX     MRESET,DATA OUT,S
00002  0 00027                       MOVE  AUX,CONTROL       RESET CONSOLE INT
                                *
00003  6 00101                       XMIT  'A',AUX
00004  6 17200                       SFL   CHAR1
00005  0 00037                       MOVE  AUX,CHAR1
00006  6 11000                       CALL  OUTCHR
00007  7 00026
00010  6 17201                       SFL   CHAR2
00011  6 00132                       XMIT  'Z',AUX
00012  0 00037                       MOVE  AUX,CHAR2
00013  6 11001                       CALL OUTCHR
00014  7 00026
00015  6 17202                       SFL  CR
00016  6 37015                       XMIT 15H,CR
00017  6 11002                       CALL OUTCHR    APPEND A CARRIAGE RETURN
00020  7 00026
00021  6 17203                       SFL  LF
00022  6 37012                       XMIT 12H,LF
00023  6 11003                       CALL OUTCHR    AND LINEFEED
00024  7 00026
00025  7 00025                       JMP  *
                                *
00026                           PROC  OUTCHR
                 200 7 0    BYTWS    RIV   200H,7,8    DUMMY WS BYTE
00026  6 07003                       SFL   DATA       IV BYTE FOR DATA IN/OUT
00027  0 37027                       MOVE  BYTWS,DATA    DATA READY FOR OUTPU
00030  6 07002                       SFL   CONTROL
00031  6 27007                       XMIT  111B,CONTROL    DATA OUT,TBRL,SFD,
00032  6 00037                       XMIT  00011111B,AUX
00033  0 00027                       MOVE  AUX,CONTROL
00034  6 00063                       XMIT  00110011B,AUX
00035  0 00027                       MOVE  AUX,CONTROL
00036  6 07003                       SFL   STATUS
00037  7 00040                       ORG   2,32
00040  4 26100                       XFC   *(TBRE)    WAIT FOR CHAR TO BE TRA
00041  6 07002                       SFL   CONTROL
```

```
55                 002 5 4    B2TO5    LIV  CONTROL,5,4    BITS 2 TO 5 OF CONTI
56   00042         6 25407             XMIT 111A,B2TO5     RESET TO DATA OUT,
57   00043         7 00044             RTN
58                                     END  OUTCHR
59
60                                     END  T2CON
```


RETURN TABLE

```
00044   4 11045
00045   7 00010
00046   7 00015
00047   7 00021
00050   7 00025
```

APPENDIX E - EXPERIMENTAL MMCMS SYSTEM PROGRAMS

```
                    0001          IDT     COMDSR
                    0002          DEF     COMDSR
                    0003  *
                    0004          REF     OTADR
                    0005          REF     OUTP
                    0006          REF     OTBYTE
                    0007          REF     INBYTE
                    0008          REF     INP
                    0009          REF     MSGSM
                    0010          REF     JCSTRT
                    0011  *
              0001  0012  ON      EQU     1
              0000  0013  OFF     EQU     0
              0000  0014  SMSIM   EQU     OFF                              TO USE MIS
                    0015  *
              0000  0016  A       EQU     0
              0001  0017  E       EQU     1
              0002  0018  X       EQU     2
              0003  0019  M       EQU     3
              0004  0020  S       EQU     4
              0005  0021  L       EQU     5
              0006  0022  B       EQU     6
              0007  0023  PC      EQU     7
              0001  0024  BR      EQU     1
                    0025  *
              00AB  0026  IDTEND  EQU     >AB     PTS AFTER LAST PGM IDT LOADED
                    0027  *
        0000  000C  0028  ERMS1   DATA    12,'NEGATIVE RECLEN FROM PRB'
        000D  1000  0029  RCLNER  @LDX    =ERMS1
        000F  D8C0  0030  MSGMR   @SSB    MSGSM
P       0011  0015  0031          DATA    MODNAM
        0012  7C00  0032          @BRU    JCSTRT
                    0033  *
              0064  0034  MXREC   EQU     100     MAXM SIZE OF MSG TEXT
P       0014        0035  SOHP    BSS     1       POINTER TO SOH, START OF MSG
        0015  C3CF  0036  MODNAM  DATA    'COMDSR'
                    0037  *ENTERS VIA BRL FROM IORMAN LIKE OTHER DSRS, M PTS TO PDT
              0018  0038  COMDSR  EQU     $
        0013  C536  0039          RMO     M,B             B PTS TO PDT, LINK TO PRB & LDT
```



```
        0019  B102  0040          DLD     2,BR    GET PRB AND LDT ADRS
        001A  8400  0041          @STA    PRBAD
        001C  C506  0042          RMO     A,B             B PTS TO PRB
        001D  8C00  0043          @STE    LDTADR  @SAVE FOR SETTING OPEN & CLOSE
        001F  C550  0044          RMO     L,A             RETURN ADR OF IORMAN
        0020  3400  0045          @STA    RETN
        0022  3400  0046          @LDA    SMBFR           STARTING ADR OF MESSAGE BUFFER
        0024  D8C0  0047          @SSB    OTADR   SET ADR REG IN MEMORY INTERFACE
                    0048  *
        0026  0103  0049          LDA     3,BR    LOCAL MESSAGE BUFFER ADR FROM PRB
        0027  807D  0050          STA     BUFAD   BUFFER ADR FROM PRB
                    0051  *GET UNIQUE # TO DIFFERENTIATE BETWEEN MSGS TO SAME DESTN
        0028  3078  0052          LDA     UNIQ    UNIQUE NUMBER
        0029  C300  0053          RIN     A,A
        002A  807B  0054          STA     UNIQ
        002B  C8C8  0055          LLA     8       SHIFT TO TOP FOR PACK
        002C  3101  0056          IOR     1,BR    COMBINE OPC (BITS 9-15) WITH MSGID
        002D  3400  0057          @STA    MSGID   SET IN MSG, MSGID & OPC PACKED
        002F  6702  0058          CPL     =2
        0030  0902  0059          LDE     2,BR    GET RECLEN FROM PRB, # CHARS
        0031  CD20  0060          SEQ             CHECK RECLEN ONLY IF A WRITE
        0032  7806  0061          BRU     ROK
        0033  CCE1  0062          SPL     E       ENSURE THAT RECLEN.GE.0
        0034  78D8  0063          BRU     RCLNER
        0035  1764  0064          LDX     =MXREC  AGREED UPON MAXM LENGTH
        0036  C412  0065          RCA     E,X     IS IT WITHIN BOUNDS?
        0037  CDC0  0066          SLE
        0038  C521  0067          RMO     X,E     ENSURE THAT ONLY MXREC CHARS ARE SENT
        0039  8879  0068  ROK     STE     RECLEN  SAVE LOCALLY FOR SEND
        003A  1717  0069          LDX     =MSL    LENGTH OF HEADER & CONTROLS
        003B  6702  0070          CPL     =2      COMPARE OPCODE TO AN ASCII WRITE
        003C  CC01  0071          SZE     E       RECLEN=0?
        003D  CD20  0072          SEQ             OR OPC.NE.WRITE?
        003E  C722  0073          RJE     X,X     THEN DON'T OUTPUT STX
        003F  0066  0074          LDA     SMBFR   GET CURRENT POSN IN SM
        0040  80D3  0075          STA     SOHP    PTR TO START OF MSG
        0041  C0A0  0076          RAD     X,A     ADD IN HEADER LENGTH
        0042  C0A0  0077          SNE             IS IT A WRITE?
        0043  C090  0078          RAD     E,A     # CHARS FROM PRB, PT PAST TEXT
```

```
0044  2701  007v          ADD   =1        +1 MORE FOR ETX CODE
0045  3800  0080          @AND  =>3FF     WRAP-A-ROUND
0047  B05E  0081          STA   SMBER     ADR IS NEXT FREE BUFFER
0049  C122  0082          RCO   X,X       -VE COUNT FOR BIX
004v  0000  0083          @LDA  =IDTEND
004B  2F03  0084          SJB   =3        PT TO START
004C  C506  0085          RMO   A,B       PTR PAST LAST IDT NAME
004D  B100  0086          DLD   0,BR
004E  A060  0087          DST   ORIGSI
004F  0102  0088          LDA   2,BR
0050  8060  0089          STA   ORIGSI+2
00v1  0000  0090          @LDA  =MSGFMT   START OF HEADER INFO
0053  C506  0091          RMO   A,B
00v4  D8C0  0092          @SSB  OUTP      OUTPUT TO SMS MEMORY
0056  3058  0093          LDA   OPC
0057  6702  0094          CPL   =2        IS OPC=WRITE?
0059  CC01  0095          SZE   E         IS RECLEN=0?
005v  CD20  0096          SEQ
005A  7806  0097          BRU   EMSG      END MESSAGE
005B  004v  0098          LDA   BUFAD     ADR OF MESSAGE TEXT
005C  C506  0099          RMO   A,B
005D  1055  0100          LDX   RECLEN    # CHARS, RECLEN FROM PRB
005E  C122  0101          RCO   X,X       -VE CHAR COUNT FOR BIX
005F  D8C0  0102          @SSB  OUTP      OUTPUT TO SMS MEMORY
0061  0703  0103  EMSG    LDA   =ETX      END OF MESSAGE TEXT
00v2  7400  0104          @BRL  OTBYTE    SEND IT
00v4  00AF  0105          LDA   SOHP
00v5  D8C0  0106          @SSB  OTADR
0067  0701  0107          LDA   =SOH
0068  7400  0108          @BRL  OTBYTE
            0109  *
            0110          IF    SMSIM,OFF,SMSI
            0111          BRU   SMS
            0112  *
      006A  0113  SMSI    EQU   $
            0114  *SMS SHOULD LOCK OUT TI SOMEWHERE IN THIS LOOP
006A  303B  0115  RRD     LDA   SMBER     POINT TO INCOMING MSG
006B  D8C0  0116          @SSB  OTADR     SET ADR REG TO ACCESS IT
006D  7400  0117          @BRL  INBYTE    READ SOH
```



```
006F  6701  0118          CPL   =SOH      SIGNIF MESSAGE
0070  CD20  0119          SEQ
0071  78F8  0120          BRU   RRD       LOOP WAITING FOR MSG TO ARRIVE
0072  17FA  0121          LDX   =-6       -VE # CHARS IN DESTN NAME
0073  0000  0122          @LDA  =DESTN
0075  C506  0123          RMO   A,B
0076  D8C0  0124          @SSB  INP       READ NAME INTO LOCAL BUFFER FOR COMPARE
0078  17FD  0125          LDX   =-3       -VE # WORDS IN NAMES
007v  027v  0126  CHKA    LDA   DESTN+3,X   GET NAME READ
007A  6237  0127          CPL   ORIGSI+3,X  COMPARE TO ORIGINAL STATION ID
007B  CD20  0128          SEQ             DOES 1 WORD MATCH?
007C  7855  0129          BRU   IGNOR     NO, MSG NOT FOR SENDER
007D  40FB  0130  CHKC    BIX   CHKA      LOOP UNTIL DONE 3 WORD NAMES
            0131  *DESTN NAME MATCHES ORIGINAL STATION ID
007E  7400  0132          @BRL  INBYTE    READ SOR
0080  17F2  0133          LDX   =-14      -VE LENGTH OF REFERENCE SECTION
0081  0000  0134          @LDA  =REFIN    REFERENCE IN
0083  C506  0135          RMO   A,B
0084  D8C0  0136          @SSB  INP       READ IN REFERENCE SECTION
0086  7400  0137          @BRL  INBYTE    READS STX OR ETX?
0088  6702  0138          CPL   =STX      IS THERE MESSAGE TEXT FOLLOWING?
008v  CD20  0139          SEQ             YES
008A  780B  0140          BRU   PSTA
008B  104C  0141          LDX   REFIN+5   GET ERRNO/RECLEN FROM MSG
008C  001A  0142          LDA   PRBAD     GET PTR TO CALLER'S PRB
008D  C506  0143          RMO   A,B
008E  9102  0144          STX   2,BR      RETURN CORRECT RECORD LENGTH
008F  C122  0145          RCO   X,X
00v0  0014  0146          LDA   BUFAD     GET BUFFER ADR FROM READ PRB SENT
00v1  C506  0147          RMO   A,B
00v2  D8C0  0148          @SSB  INP       READ TEXT INTO CALLER'S BUFFER
0094  7400  0149          @BRL  INBYTE    READ ETX. INTERFACE REG PTS TO NEXT
      00v6  0150  PSTA    EQU   $
00v6  0060  0151          LDA   REFIN+4   GET STATUS BITS
00v7  C8C8  0152          LLA   8         SHIFT TO TOP BYTE
00v8  340E  0153          IOR   *PRBAD    COMBINE WITH LUN, PRB(0)
009v  340D  0154          STA   *PRBAD    SET FLAGS IN PRB
            0155  *CHECK BIT 1 TO SEE IF WE HAVE TO SET ERRNO IN PDT
009A  DB11  0156          TABO  1         CHECK FOR ERROR
```

```
 009B  7B1A  0157          BRU   INXT     NO, CHECK FOR AN OPEN OR CLOSE
 009C  0808  0158          LDE   LDTADR   PTR TO LDT
 009D  C516  0159          RMO   E,B
 009E  0902  0160          LDE   2,BR     PTS TO PDT NOW
 009F  C516  0161          RMO   E,B
 00A0  0057  0162          LDA   REFIN+5  GET ERRNO/RECLEN FROM MSG
 00A1  0868  0163          LRL   8        SHIFT DOWN
 00A2  8104  0164          STA   4,BR     SET ERROR IN PDT
 00A3  7822  0165          BRU   INCA     INCR LOCAL MSG ADR TO MATCH INTFCE
 00A4  0000  0166  UNIQ    DATA  0
 00A5  0000  0167  BUFAD   DATA  0        LOCAL STORAGE FOR START OF BUFR FROM PRB
 00A6  0100  0168  SMBFR   DATA  >100
 00A7  0000  0169  PRBAD   DATA  0
 00A8  0000  0170  LDTADR  DATA  0
P 00A9        0171  HEIN    BSS   1
             0172  *
       0001  0173  SOH     EQU   >01        START OF HEADER CODE
       001E  0174  SOR     EQU   >1E        START OF REFERENCE CODE
       0002  0175  STX     EQU   >02        START OF MESSAGE TEXT CODE
       0003  0176  ETX     EQU   >03        END OF MESSAGE TEXT CODE
             0177  DATB    FAM   8,8
             0178  *
       0017  0179  MSL     EQU   23         # CHARS IN HEADER WITH CONTROL
       00AA  0180  MSGFMT  EQU   $
 00AA  00CC  0181          DATB  0,'L'      NO SOH
 00AB  CFC7  0182          DATB  'O','G'
 00AC  A0A0  0183          DATB  ' ',' '
 00AD  A01E  0184          DATB  ' ',SOR
 00AE  0000  0185  RSI     DATA  0          REFERENCE STATION ID, ERROR HANDLING IN COMM PROC
 00AF  00C7  0186  ORIGSI  DATA  'PGMNAM'
 00B2  0000  0187  MSGID   DATB  0,0
       0082  0188  OPC     EQU   MSGID      OPCODE FROM PRB
 00B3  0000  0189  RECLEN  DATA  0          RECORD LENGTH, # CHARS FROM PRB
 00B4  0000  0190  CHKSUM  DATA  0
 00B5  0200  0191          DATB  STX
             0192  *
             0193  *CHECK BIT 5 TO SEE IF WE HAVE TO SET FLAGS IN LDT
             0194  *SINCE ERROR BIT WAS NOT SET, OPC<11
 00B6  00F8  0195  INXT    LDA   OPC        GET MSGID/OPC
```



```
 00B7  6707  0196          CPL   =7
 00B8  C0C0  0197          SLE                    IS OPC BETWEEN 7411? OPEN/CLOSE?
 00B9  780C  0198          BRU   INCA             NO
 00BA  0709  0199          CPL   =9
 00BB  CD40  0200          SGT                    IS IT A CLOSE?
 00BC  7804  0201          BRU   CL               YES
 00BD  DB53  0202  OP      SABO  3                SET OPEN BIT
 00BE  0839  0203          LDE   REFIN+5          GET RECLEN FROM MSG
 00BF  8903  0204          STE   3,BR             STORE IN LDT
 00C0  7801  0205          BRU   $+2              SKIP
 00C1  DB43  0206  CL      SABZ  3                ZERO OPEN BIT ON CLOSE
 00C2  8100  0207          STA   0,BR             SET IN LDT
 00C3  0100  0208          LDA   0,BR             GET FLAGS FROM LDT
 00C4  00E3  0209          LDA   LDTADR           PTR TO LDT
 00C5  C506  0210          RMO   A,B              GIVES ACCESS INTO IT
 00C6  00EB  0211  INCA    LDA   OPC              GET MSGID & OPCODE PACKED
 00C7  6700  0212          CPL   =0               IS OPC A READ IMPLYING MSG TEXT?
 00C8  0000  0213          LDA   SMBFR            GET CURRENT POSN IN SM
 00C9  2717  0214          ADD   =MSL             ADD IN HEADER LENGTH
 00CA  CD20  0215          SEQ
 00CB  7802  0216          BRU   $+3              CONTROL
 00CC  2028  0217          ADD   REFIN+5          TEXT, ADDIN RECORD LENGTH
 00CD  2701  0218          ADD   =1               +1 FOR ETX
 00CE  3400  0219          @AND  =>3FF            WRAPAROUND
 00CF  80D5  0220          STA   SMBFR            UPDATE ADR OF NEXT MSG
 00D1  7CD7  0221          BRU   *REIN
             0222  *
 00D2  0020  0223  IGNOR   STX   REFIN            SAVE IT
 00D3  1000  0224          @LDX  =ERMS2
 00D5  38C0  0225          @SSB  MSGSM
P 00D7  0015  0226          DATA  MODNAM
 00D8  101A  0227          LDX   REFIN
 00D9  78A3  0228          BRU   CHKC
 00DA  0018  0229  ERMS2   DATA  24,'DESTN FROM SMS MESSAGE IS NOT COORDINATOR-'
P 00F0        0230  DESTN   BSS   3
P 00F3        0231  REFIN   BSS   7
             0232  *
             0233          IF    SMSIM,OFF,SM52
             0234  *FOR TIME BEING SIMULATE SMS RESPONSES
```

```
0235  SMS     EQU   $
0236          LDA   MSGID     GET PACKED MSGID & OPC
0237          CPL   =0        IS OPC A HEAD?
0238          SEQ             YES
0239          BRU   MSG2      NO, THEN A CONTROL MSG RETURNED
0240          LDX   =-81      TOTAL LENGTH OF MSG FROM SMS
0241          @LDA  =MSGSM5   START OF MSG
0242          RMO   A,B       SET REGS FOR OUTP
0243          BRU   OUTIT
0244  MSG2    LDX   =-23      TOTAL LENGTH OF CONTROL MSG.
0245          @LDA  =MSGSM2   STARTING ADR
0246          RMO   A,B
0247  OUTIT   EQU   $
0248          @LDA  =>FF      MASK FOR RIGHT BYTE
0249          AND   B,BR      ISOLATE STATUS BITS
0250          RMO   A,E       HOLD ONTO STATUS
0251          LDA   UNIQ      MSGID ON MSG SENT
0252          LLA   8         SHIFT INTO TOP OF WORD
0253          ROR   E,A       COMBINE MSGID & STATUS
0254          STA   B,BR
0255          @SSB  OUTP      OUTPUT MSG EXACTLY AS IS
0256          BRU   RRD       NOTIFY TI THAT THERE IS A MSG
0257  *
0258  MSGSMS  DATB  SOH,'P'
0259          DATB  'O','M'
0260          DATB  'N','A'
0261          DATB  'M',SOR
0262          DATA  0
0263          DATA  'LOG '
0264          DATB  2,0       MSGID,STATUS RETURNED BY LOGDSR
0265          DATA  57
0266          DATA  0         CHECKSUM
0267          DATB  STX,'T'
0268          DATA  'HIS MESSAGE SIMULATES SMS RESPONSE TO KEYIN FROM CONSOLE'
0269          DATB  ETX
0270  *
0271  MSGSM2  DATB  SOH,'P'
0272          DATB  'O','M'
0273          DATB  'N','A'
```



```
0274          DATB  'M',SOR
0275          DATA  0
0276          DATA  'LOG '
0277          DATB  0,>20     MSGID,STATUS  EOF BIT SET
0278          DATA  25        DUMMY RECLEN
0279          DATA  0         CHECKSUM
0280          DATB  ETX
00FA  0281  SMS2  EQU   $
0000  0282          END
```

A	0000	B	0006	BR	0001	BUFAD	00A5
CHKA	007A	CHKC	007D	R CHKSUM	0084	CL	00C1
COMDSR	0018	DATB	8080	DESTN	00F0	E	0001
EMSG	0061	ERMS1	0000	ERMS2	00DA	ETX	0003
IDTEND	00A8	IGNOR	00D2	INBYTE	0003	INCA	00C6
INP	0004	JCSTRT	0005	L	0005	LDTADR	00A8
M	0003	MODNAM	0015	MSGFMT	00AA	MSGID	00B2
MSGSM	0005	R MSGWR	000F	MSL	0017	MXREC	0064
OFF	0000	R ON	0001	R OP	00BD	OPC	00B2
ORIGS1	00AF	OTADR	0000	OTBYTE	0002	OUTP	0001
R PC	0007	PRBAD	00A7	PSTA	0096	RCLNER	000D
RECLEN	0033	REFIN	00F3	RETN	00A9	ROK	0039
RRJ	00BA	R RSI	00AE	R S	0004	SMBFR	00A6
R SMS1	00BA	R SMS2	00FA	SMSIM	0000	SOH	0001
SOHP	0014	SOR	001E	STX	0002	TNXT	00B6
UNIO	00A4	X	0002				

0000 ERRORS IN COMDSR

```
                                PROG  TLOG
                                LIST  I
000000          DBUG    SET   0
                *MAINLINE TESTS LOGDSR AND UTILITY SUBROUTINES
                *WITH SM INITIALIZED BY TI, SHOULD BE ABLE TO HA
                *INCLUDES OUTPUTTING A MESSAGE TEXT FROM SM TO C
                *READING A LINE FROM CONSOLE AND PASSING IT TO T
                *
                *IV BYTES USED BY MORE THAN 1 ROUTINE
001 7 0         DISPLAY LIV   1,7,8           LED DISPLAY AND SWI
003 7 0         LSADR   LIV   3,7,8           LEAST SIGNIF SM ADR
004 7 0         CONTROL LIV   4,7,8           SM CONTROL BYTE
004 7 5         MSADR   LIV   CONTROL,7,5     MOST SIGNIF SM ADR
004 0 1         HALTI   LIV   CONTROL,0,1     FOR LOCKING OUT TI
005 7 0         CNTRL   LIV   5,7,8           CONTROL FOR CONSOLE
                *
000314          L       EQU   314H
000317          O       EQU   317H
000307          G       EQU   307H
000240          BLNK    EQU   240H
000212          LF      EQU   212H
000215          CR      EQU   215H
                *
000144          MXREC   EQU   100             # CHARS ALLOWED PER
000115          SCRWID  EQU   77              # CHARS ON LINE OF
000030          LNSC    EQU   24              # LINES DISPLAYABLE
000001          SOH     EQU   01              START OF HEADER COD
000036          SOR     EQU   36H             START OF REFERENCE
000002          STX     EQU   02              START OF MESSAGE TE
000003          ETX     EQU   03              END OF MESSAGE TEXT
                *
                *
177 7 1         PSR     RIV   177H,7,1        PAGE SELECT REG
000000          PAGE0   EQU   0
000001          PAGE1   EQU   1
200 7 0         BYTWS   RIV   200H,7,8
200 0 1         BIT0    RIV   BYTWS,0,1
200 1 1         BIT1    RIV   BYTWS,1,1
200 2 1         BIT2    RIV   BYTWS,2,1
200 3 1         BIT3    RIV   BYTWS,3,1
200 4 1         BIT4    RIV   BYTWS,4,1
200 5 1         BIT5    RIV   BYTWS,5,1
200 6 1         BIT6    RIV   BYTWS,6,1
200 7 1         BIT7    RIV   BYTWS,7,1
                *
                *NOP IS 1 INSTRUCTION SO GUARANTEES A 300 NS DELA
                NOP     MACRO
                        MOVE  AUX,AUX
                        ENDM
                *
                BNE     MACRO D,CHR,ADR
                        SEL   DISPLAY
                        MOVE  D,AUX
                        MOVE  AUX,DISPLAY
                        XMTT  CHR,AUX
                        XOR   D,AUX
                        NZT   AUX,ADR
                        ENDM
```

```
                                *
                        BEQ     MACRO D,CHR,ADR
                                XMIT  CHR,AUX
                                XOR   D,AUX
                                NZT   AUX,*+2
                                JMP   ADR
                                ENDM
                                *
                200 7 0 MSG     RIV   200H,7,8        MESSAGE BLOCK
                200 7 0 RSI     RIV   MSG,7,8
                201 7 0 RSI1    RIV   RSI+1,7,8
                202 7 0 ORIGSI  RIV   MSG+2,7,8       6 CHAR ORIGINATOR N
                203 7 0 ORIG1   RIV   ORIGSI+1,7,8
                204 7 0 ORIG2   RIV   ORIGSI+2,7,8
                205 7 0 ORIG3   RIV   ORIGSI+3,7,8
                206 7 0 ORIG4   RIV   ORIGSI+4,7,8
                207 7 0 ORIG5   RIV   ORIGSI+5,7,8
                210 7 0 MSGID   RIV   MSG+8,7,8
                211 7 0 OPC     RIV   MSG+9,7,8
                212 7 0 DUM     RIV   MSG+10,7,8      RECLEN IS 2 BYTES O
                213 7 0 RECLEN  RIV   MSG+11,7,8
                214 7 0 CHKSUM  RIV   MSG+12,7,8
                215 7 0 CHKS1   RIV   MSG+13,7,8
                 000200 REF     EQU   RSI
                216 7 0 SMSG    RIV   MSG+14,7,8      RESERVE NEXT MXREC
                 000216 MSGTEXT EQU   SMSG
                                *
                363 7 0 EMSG    RIV   SMSG+MXREC+1,7,8  1ST WORD AFTER M
                363 7 0 PMSGL   RIV   FMSG,7,8        PHYSICAL RECORD SIZ
                364 7 0 SPARE1  RIV   FMSG+1,7,8
                365 7 0 MRECL   RIV   FMSG+2,7,8      MAXIM RECLEN
                366 7 0 ERRNO   RIV   FMSG+3,7,8      ERROR # TO BE RETUR
                367 7 0 LNCNT   RIV   EMSG+4,7,8
                370 7 0 STAT    RIV   FMSG+5,7,8      STATUS TO BE RETURN
                370 5 1 OPCL    RIV   STAT,5,1        1 WHEN DEVICE HAS B
                370 6 1 PGS     RIV   STAT,6,1        1 WHEN PAGEMODE IS
                370 7 1 STP     RIV   STAT,7,1        1 WHEN STOPC HAS BE
                371 7 0 BELL    RIV   FMSG+6,7,8
                372 7 0 PROMPT  RIV   FMSG+7,7,8
                373 7 0 SPARE2  RIV   FMSG+8,7,8
                374 7 0 RETN    RIV   FMSG+9,7,8
                375 7 0 LSADR2  RIV   FMSG+10,7,8
                376 7 5 MSADR2  RIV   FMSG+11,7,5
                                *
                 000000 SMSCP   EQU   *
00000  6 17371                  SEL   BELL
00001  6 00207                  XMIT  207H,AUX
00002  0 00037                  MOVE  AUX,BELL
00003  6 17372                  SEL   PROMPT
00004  6 00276                  XMIT  276H,AUX
00005  0 00037                  MOVE  AUX,PROMPT
00006  6 07003                  SEL   LSADR           SM ADR CONSISTS OF
00007  6 27000                  XMIT  0,LSADR         SET INITIAL ADR TO
00010  6 07004                  SEL   MSADR
00011  6 27501                  XMIT  1,MSADR
00012  6 17367                  SEL   LNCNT
00013  6 37030                  XMIT  LNSC,LNCNT
00014  6 17370                  SEL   STAT
```

```
117  00015  6 37302          XMIT  010R,STAT,3    SET DEFAULTS FOR OP
118                  *
119  00016  6 17363  RLOOP   SEL   PMSGL          PHYSICAL MSG RECORD
120  00017  6 00144          XMIT  MXREC,AUX
121  00020  0 00037          MOVE  AUX,PMSGL
122  00021  6 11000          CALL  INSOH          ONLY INPUT 1 CHAR F
     00022  7 01133
123                          BNF   R1,SOH,*-4     HAVE TO FIND SOH
124  00031  6 17374          SEI   RETN
125  00032  6 00240          XMIT  BLNK,AUX
126  00033  0 00037          MOVE  AUX,RETN
127  00034  6 11001          CALL  OTBYTE
     00035  7 01277
128                  *
129  00036  6 11002          CALL  INBYTE         READ 1ST CHAR OF DE
     00037  7 01266
130                          BNF   R1,L,*
131  00046  6 11003          CALL  INBYTE
     00047  7 01266
132                          BNF   R1,O,*
133  00056  6 11004          CALL  INBYTE
     00057  7 01266
134                          BNF   R1,G,*
135  00066  6 11005          CALL  INBYTE
     00067  7 01266
136                          BNF   R1,BLNK,*
137  00076  6 11006          CALL  INBYTE
     00077  7 01266
138                          BNF   R1,BLNK,*
139  00106  6 11007          CALL  INBYTE
     00107  7 01266
140                          BNF   R1,BLNK,*
141  00116  6 11010          CALL  INBYTE         READ 1 CHAR, SHOULD
     00117  7 01266
142                          BNF   R1,SOR,*
143  00126  6 03362          XMIT  -14,R3         -VE # CHARS IN REF
144  00127  6 06200          XMIT  REF,R6         WHERE WE WANT TO ST
145  00130  0 06017  LOOP1   MOVE  R6,IVR         ACCESS TO WORKING S
146  00131  6 11011          CALL  INBYTE         GET 1 CHAR FROM SM
     00132  7 01266
147  00133  0 01037          MOVE  R1,BYTWS       STORE IN WS
148  00134  6 00001          XMIT  1,AUX          INCR VALUE
149  00135  1 06006          ADD   R6,R6          MOVE PTR FORWARD
150  00136  1 03003          ADD   R3,R3          BYTE COUNTER
151  00137  5 03130          NZT   R3,LOOP1       COPY REF SECTION TO
152  00140  6 11012          CALL  INBYTE         SHOULD BE EITHER STX
     00141  7 01266
153                          BNF   R1,STX,CETX
154                  *STX, IMPLIES THAT THERE IS A MESSAGE TEXT
155  00150  6 06216          XMIT  MSGTEXT,R6     WHERE TO STORE TEXT
156                  *SET R3 TO CONTAIN -VE(MIN(RECLEN FROM MSG,PHYS.
157  00151  6 17213          SEL   RECLEN         GET # CHARS TO BE O
158  00152  6 00377          XMIT  -1,AUX         MAKE IT -VE FOR COU
159  00153  3 37003          XOR   RECLEN,R3      1'S COMPLEMENT
160  00154  6 00001          XMIT  1,AUX          2'S COMP
161  00155  1 03003          ADD   R3,R3          SET UP -VE # CHARS
162  00156  6 17363          SEL   PMSGL          GET PHYS MSGLEN
163  00157  0 37000          MOVE  PMSGL,AUX      SET UP FOR COMPARE
```

```
 164  00160  1 03037          ADD   R3,PMSGL
 165  00161  5 30123          NZT   BIT0,*+2          ENSURE THAT MSGLEN
 166  00162  7 00164          JMP   LOOP2
 167  00163  6 03234          XMIT  -MXREC,R3         -VE MAXM MSG LENGT
 168                          *
 169  00164  6 11013  LOOP2   CALL  INBYTE           GET 1 BYTE FROM SM
      00165  7 01266
 170  00166  0 06017          MOVE  R6,IVR            ACCESS TO WS
 171  00167  0 01037          MOVE  R1,BYTWS          STORE IN WS
 172  00170  6 00001          XMIT  1,AUX             INCR
 173  00171  1 06006          ADD   R6,R6
 174  00172  1 03003          ADD   R3,R3
 175  00173  5 03164          NZT   R3,LOOP2          COPY MESSAGE TEXT
 176  00174  6 11014          CALL  INBYTE            READ NEXT CHAR
      00175  7 01266
 177                  CETX    BNE   R1,ETX,*          SHOULD BE ETX NOW
 178  00204  6 11015          CALL  LOGDSR            ROUTE TO HANDLING
      00205  7 00437
 179                          *RETURNS FROM LOGDSR, STAT WILL BE SET, MAYBE EI
 180                          *RECLEN MAY HAVE BEEN REDUCED
 181                          *RETURN A CONTROL MESSAGE
 182                          *
 183                          *       SEL   CONTROL
 184                          *       XMIT  0,HALTI           LOCKOUT TI WHILE SE
 185                          *SAVE ADR FOR SOH TO BE SENT AFTER MSG IS COMPLE
 186  00206  6 17375          SEL   LSADR2
 187  00207  6 07003          SEL   LSADR
 188  00210  0 27037          MOVE  LSADR,LSADR2
 189  00211  6 07004          SEL   MSADR
 190  00212  0 27500          MOVE  MSADR,AUX
 191  00213  6 17376          SEL   MSADR2
 192  00214  0 00537          MOVE  AUX,MSADR2
 193  00215  6 11016          CALL  AINC              INCR TO NEXT POSN I
      00216  7 01307
 194  00217  6 17202          SEL   ORIGSI
 195  00220  6 06202          XMIT  ORIGSI,R6         SEND MSG TO PGNAM O
 196  00221  6 03372          XMIT  -6,R3
 197  00222  0 06017  LOOP4   MOVE  R6,IVR
 198  00223  6 11017          CALL  OTBYTE
      00224  7 01277
 199  00225  6 00001          XMIT  1,AUX
 200  00226  1 06006          ADD   R6,R6
 201  00227  1 03003          ADD   R3,R3
 202  00230  5 03222          NZT   R3,LOOP4
 203  00231  6 17374          SEL   RETN              AFTER LOGDSR IT'S FI
 204  00232  6 37036          XMIT  SOR,RETN          SET UP REFERENCE SE
 205  00233  6 11020          CALL  OTBYTE
      00234  7 01277
 206  00235  6 17374          SEL   RETN
 207  00236  6 37000          XMIT  0,RETN            ZERO FOR RSI WORD
 208  00237  6 11021          CALL  OTBYTE
      00240  7 01277
 209  00241  6 11022          CALL  OTBYTE
      00242  7 01277
 210  00243  6 00314          XMIT  1,AUX             SET ORIG TO LOG
 211  00244  0 00037          MOVE  AUX,RETN
 212  00245  6 11023          CALL  OTBYTE
      00246  7 01277
```

```
213  00247  6 00317.        XMIT  0,AUX
214  00250  0 00037         MOVE  AUX,RETN
215  00251  6 11024         CALL  OTBYTE
     00252  7 01277
216  00253  6 00307         XMIT  6,AUX
217  00254  0 00037         MOVE  AUX,RETN
218  00255  6 11025         CALL  OTBYTE
     00256  7 01277
219  00257  6 00240         XMIT  BLNK,AUX
220  00260  0 00037         MOVE  AUX,RETN
221  00261  6 11026         CALL  OTBYTE
     00262  7 01277
222  00263  6 11027         CALL  OTBYTE
     00264  7 01277
223  00265  6 11030         CALL  OTBYTE
     00266  7 01277
224  00267  6 17210         SEL   MSGID
225  00270  6 11031         CALL  OTBYTE
     00271  7 01277
226  00272  6 17370         SEL   STAT
227  00273  6 11032         CALL  OTBYTE
     00274  7 01277
228  00275  6 17366         SEL   ERRNO
229  00276  6 11033         CALL  OTBYTE          RETURNED IN TOP HAL
     00277  7 01277
230  00300  6 17213         SEL   RECLEN
231  00301  6 11034         CALL  OTBYTE
     00302  7 01277
232  00303  6 17374         SEL   RETN
233  00304  6 37000         XMIT  0,RETN
234  00305  6 11035         CALL  OTBYTE          ZERO FOR CHECKSUM W
     00306  7 01277
235  00307  6 11036         CALL  OTBYTE
     00310  7 01277
236  00311  6 17370         SEL   STAT            IF ERRORS THEN NO T
237  00312  7 00340         ORG   28,32
238  00340  5 31131         NZT   BIT1,ENDT
239  00341  6 17213         SEL   RECLEN          CHECK IF WE GOT ANY
240  00342  5 37004         NZT   RECLEN,*+2
241  00343  7 00371         JMP   ENDT            NO
242  00344  6 17211         SEL   OPC             CHECK FOR A READ TO
243  00345  6 00013         XMIT  11,AUX
244  00346  1 37037         ADD   OPC,OPC
245                         ORG   25,32
246  00347  5 37031         NZT   OPC,ENDT
247  00350  6 17374         SEL   RETN
248  00351  6 37002         XMIT  STX,RETN
249  00352  6 11037         CALL  OTBYTE
     00353  7 01277
250  00354  6 06216         XMIT  MSGTEXT,R6
251  00355  6 17213         SEL   RECLEN
252  00356  6 00377         XMIT  -1,AUX
253  00357  3 37003         XOR   RECLEN,R3
254  00360  6 00001         XMIT  1,AUX
255  00361  1 03003         ADD   R3,R3
256  00362  0 06017  LOOP5  MOVE  R6,IVR
257  00363  6 11040         CALL  OTBYTE
     00364  7 01277
```

```
585  01107  5 04111            NZT   R4,*+2
586  01110  7 00615            JMP   RRET          NO CHARACTERS KEYED
587  01111  6 00377            XMIT  -1,AUX        DECR VALUE
588  01112  1 06006            ADD   R6,R6         BACKUP PTR TO LAST
589  01113  0 06017            MOVE  R6,IVR
590                            ORG   7,32
591                   STRIPB   BNE   BYTWS,BLNK,CCNT
592  01122  6 00377            XMIT  -1,AUX        DECR VALUE
593  01123  1 06006            ADD   R6,R6         DECR PTR
594  01124  1 04004            ADD   R4,R4         DECR COUNT
595  01125  5 04114            NZT   R4,STRIPB     CONTINUE TO STRIP
596  01126  6 17213   CCNT     SEL   RECLEN
597  01127  0 04037            MOVE  R4,RECLEN
598  01130  6 17374            SEL   RETN
599  01131  0 37011            MOVE  RETN,R11
600  01132  7 01400            RTN
601                   *
602                            END   LOGDSR
```

```
604  01133                     PROC  INSOH
605          002 7 0  DATA     LIV   2,7,8         DATA BYTE FOR SM
606          004 2 1  RW       LIV   CONTROL,2,1
607          004 1 1  MEMSTR   LIV   CONTROL,1,1
608           000001  READ     EQU   1
609                   *
610  01133  6 07004            SEL   CONTROL
611  01134  6 22101            XMIT  READ,RW
612                            NOP
613  01136  6 21100            XMIT  0,MEMSTR
614                            NOP
615  01140  6 21101            XMIT  1,MEMSTR
616  01141  6 07002            SEL   DATA
617  01142  0 27001            MOVE  DATA,R1
618  01143  7 01400            RTN
619                            END   INSOH
```

```
 621  01144                          PROC, IOCHR
 622          005 3 1    TBRL       LIV   CNTRL,3,1
 623          006 7 0    DATA       LIV   6,7,8          DATA BYTE TO/FROM C
 624                     *STATUS SHARES IV BYTE WITH DATA
 625          006 7 0    STATUS     LIV   6,7,8          STATUS RECEIVED FRO
 626          006 5 3    ERRS       LIV   STATUS,5,3     PE, FE, OE ARE BITS
 627          006 6 1    TBRE       LIV   STATUS,6,1     XMIT BUFFER EMPTY, I
 628          006 7 1    DR         LIV   STATUS,7,1     DATA READY, BIT 7
 629                     *
 630          000233     ESC        EQU   233H
 631                     *ON ENTRY IVR PTS TO TEXT IN WS, R3 CONTAINS -VE
 632                                ENTRY OUTCHR
 633  01144  6 07006               SEL   DATA             IV BYTE FOR DATA IN
 634  01145  0 37027               MOVE  BYTWS,DATA       DATA READY FOR OUTPU
 635  01146  6 07005               SEL   CNTRL
 636  01147  6 27007               XMIT  111B,CNTRL       DATA OUT,TBRL,SFD,RF
 637  01150  6 00037               XMIT  00011111B,AUX    BRING TBRL HIGH
 638  01151  0 00027               MOVE  AUX,CNTRL
 639  01152  6 00063               XMIT  00110011B,AUX    DATA IN,READ STATUS
 640  01153  0 00027               MOVE  AUX,CNTRL
 641  01154  6 07006               SEL   STATUS
 642                                ORG   10,32
 643  01155  4 26115               XEC   *(TBRE)          WAIT FOR CHAR TO BE
 644  01156  5 27122               NZT   DR,KEYIN         CHAR KEYED IN?
 645  01157  6 07005    ORET       SEL   CNTRL
 646          005 5 4    B2TO5      LIV   CNTRL,5,4       BITS 2 TO 5 OF CNTRL
 647  01160  6 25407               XMIT  111B,B2TO5       SET BACK TO DATA OU
 648  01161  7 01400               RTN
 649                     *
 650  01162  0 11005    KEYIN      MOVE  R11,R5           SAVE RETURN
 651  01163  6 11070               CALL  RDCHR            ENTRY PT TO INPUT A
      01164  7 01243
 652  01165  0 05011               MOVE  R5,R11
 653                               BNF   R1,ESC,CSTX      ESC-STOP OUTPUTTING
 654  01174  6 00215               XMIT  CR,AUX
 655  01175  0 00037               MOVE  AUX,BYTWS
 656  01176  6 11071               CALL  OUTCHR
      01177  7 01144
 657  01200  6 00212               XMIT  LF,AUX
 658  01201  0 00037               MOVE  AUX,BYTWS
 659  01202  6 11072               CALL  OUTCHR
      01203  7 01144
 660                     *SET IGNORF BIT IN STAT AND SET UP RETURN OUT OF
 661  01204  6 17370               SEL   STAT
 662  01205  6 33101               XMIT  1,BIT3
 663  01206  6 17374               SEL   RETN
 664  01207  0 37011               MOVE  RETN,R11
 665  01210  7 01157               JMP   ORET
 666                     *
 667                     CSTX       BNF   R1,STX,CETX
 668  01217  6 17370               SEL   PGS
 669  01220  6 36101               XMIT  1,PGS
 670  01221  6 17367               SEL   LNCNT
 671  01222  6 37030               XMIT  LNSC,LNCNT
 672  01223  0 06017               MOVE  R6,IVR
 673  01224  7 01157               JMP   ORET
 674                     CETX       BNF   R1,ETX,ORET
 675  01233  6 17370               SEL   PGS
```

```
676  01234  6 36100          XMIT  0,PGS
677  01235  7 01157          JMP   ORET
678                          ENTRY INCHR
679  01236  6 07005  RRD     SEL   CNTRL
680  01237  6 00063          XMIT  00110011B,AUX   DATA IN,RRD
681  01240  0 00027          MOVE  AUX,CNTRL       READ STATUS
682  01241  6 07006          SEL   STATUS
683  01242  4 27102          XEC   *(DR)           WAIT FOR CHAR TO BE
684                          *
685                          ENTRY RDCHR
686                          ORG   23,32
687  01243  5 25313          NZT   ERRS,ERHAN      CHECK FOR ERRORS
688  01244  6 07005          SEL   CNTRL
689  01245  6 00070          XMIT  00111000B,AUX   SFD,DATA IN,DRR
690  01246  0 00027          MOVE  AUX,CNTRL       READ DATA
691  01247  6 07006          SEL   DATA            ACCESS TO CHAR
692                          NOP                   ENSURE 500 NS
693  01251  0 27001          MOVE  DATA,R1         GET CHAR ENTERED
694  01252  7 01400  IORET   RTN
695                          *
696  01253  6 17366  ERHAN   SEL   ERRNO
697  01254  6 00100          XMIT  100H,AUX        CODE FOR ERROR
698  01255  3 25300          XOR   ERRS,AUX
699  01256  0 00037          MOVE  AUX,ERRNO
700  01257  6 07001          SEL   DISPLAY
701  01260  0 37027          MOVE  ERRNO,DISPLAY   FOR TIME BEING DISP
702  01261  6 17370          SEL   STAT
703  01262  6 31101          XMIT  1,BIT1          SET ERROR BIT
704  01263  6 17374          SEL   RETN            GET OUT OF LOGDSR
705  01264  0 37011          MOVE  RETN,R11
706  01265  7 01400          RTN
707                          END   IOCHR
```

```
709  01266                        PROC  IOBYTE
710                          *IMPLEMENTS AUTO INCREMENT AND WRAPAROUND ON SHA
711                          *USING MEMORY AS A CIRCULAR QUEUE PROVIDES A REC
712                          *MESSAGE TRANSFER
713          002 7 0    DATA     LIV   2,7,8          IV BYTE 2
714          004 5 1    MSOVF    LIV   MSADR,5,1
715          004 2 1    RW       LIV   CONTROL,2,1
716          004 1 1    MEMSTR   LIV   CONTROL,1,1
717           000001    READ     EQU   1
718           000000    WRIT     EQU   0
719                          *
720                          ENTRY.INBYTE
721  01266   6 07004           SEL   CONTROL
722  01267   6 22101           XMIT  READ,RW
723                            NOP
724  01271   6 21100           XMIT  0,MEMSTR
725                            NOP
726  01273   6 21101           XMIT  1,MEMSTR
727  01274   6 07002           SEL   DATA
728  01275   0 27001           MOVE  DATA,R1        GET BYTE READ
729  01276   7 01307           JMP   AINC2          INCR ADR
730                          *
731                          ENTRY OTBYTE
732  01277   6 07004           SEL   CONTROL
733  01300   6 22100           XMIT  WRIT,RW
734  01301   6 07002           SEL   DATA
735  01302   0 37027           MOVE  BYTWS,DATA
736  01303   6 07004           SEL   CONTROL
737  01304   6 21100           XMIT  0,MEMSTR
738                            NOP
739  01306   6 21101           XMIT  1,MEMSTR
740                          *
741                          ENTRY AINC
742  01307   6 00001    AINC2  XMIT  1,AUX          INCR FOR ADR
743  01310   6 07003           SEL   LSADR
744  01311   1 27027           ADD   LSADR,LSADR
745                            ORG   3,32
746  01312   5 10314           NZT   OVF,*+2        ADR IS IN 2 HALVES
747  01313   7 01400           RTN
748  01314   6 07004           SEL   MSADR
749  01315   1 27527           ADD   MSADR,MSADR
750                            ORG   3,32
751  01316   5 25120           NZT   MSOVF,*+2      WRAPAROUND?
752  01317   7 01400           RTN
753  01320   6 27500           XMIT  0,MSADR
754  01321   7 01400           RTN
755                            END   IOBYTE
756                          *
757                            END   TLOG
```

RETURN TABLE

```
01400   4 11001
01401   7 00023
01402   7 00036
01403   7 00040
01404   7 00050
```